



**Escuela Superior
de Ingeniería y Tecnología**
Universidad de La Laguna

Trabajo de Fin de Grado

**Prodef-SaaS: Despliegue y puesta en
marcha de un servicio para la resolución
de problemas de optimización**

*Prodef-SaaS: Deployment and start-up of a service for
optimization problem solving*

Ángel Tornero Hernández

La Laguna, 8 de julio de 2022

D^a. **Gara Miranda Valladares**, con N.I.F. 78.563.584-T, profesora Titular de Universidad adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora

D. **Andrés Calimero García Pérez**, con N.I.F. 51.149.384-Y, Graduado en Ingeniería Informática, como cotutor

C E R T I F I C A (N)

Que la presente memoria titulada:

“Prodef-SaaS: Despliegue y puesta en marcha de un servicio para la resolución de problemas de optimización”

ha sido realizada bajo su dirección por D. **Ángel Tornero Hernández**, con N.I.F. 45.939.001-C.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 8 de julio de 2022

Agradecimientos

En primer lugar, me gustaría agradecer a mi tutora, Gara; y a mi cotutor, Andrés, por haber compartido tanto conocimiento conmigo además de haberme apoyado en todo momento durante la realización de este Trabajo de Fin de Grado.

A continuación, quiero agradecer a mi familia porque siempre me han dado apoyo y cariño, y me han aportado todos los medios necesarios para que yo cumpla mis objetivos.

También me gustaría agradecer al profesor José Manuel Gálvez Lamolda, por haber demostrado en cada asignatura que imparte que su máxima prioridad es que aprendamos de verdad y, sobretodo, que nos vaya bien en el futuro.

Por último, me gustaría agradecer a los amigos que hice en la carrera porque sin ellos, esta etapa de mi vida no habría estado tan llena de buenos momentos. En especial a Gabriel García Jaubert, que me ha ayudado, apoyado y acompañado a lo largo de todo este camino y fue más que “un amigo de la Universidad” desde el primer momento.

Licencia



© Esta obra está bajo una licencia de Creative Commons Reconocimiento-
NoComercial-CompartirIgual 4.0 Internacional.

Resumen

En la actualidad, los algoritmos de optimización bio-inspirados son ampliamente utilizados en ámbitos académicos y existe una gran cantidad de investigación dedicada a estos. No obstante, son métodos altamente complejos con una larga curva de aprendizaje, no solo de comprensión sino también de implementación. Es por esto que, a pesar de ser técnicas considerablemente útiles, su uso no se extiende mucho más allá del ámbito investigador. Prodef es una herramienta que surge en este contexto con el objetivo de expandir el alcance de estas técnicas, tratando de acercarlas a entornos empresariales. Prodef permite modelar problemas con una interfaz gráfica basada en bloques, definir algoritmos sin escribir código y generar instancias de problemas para su posterior resolución. Sin embargo, en un principio Prodef no era una herramienta que tuviera fácil acceso: utilizarla requería instalar manualmente múltiples dependencias, y se debía hacer uso de una terminal de Linux para poner en marcha todos los microservicios. Para complicar aún más las cosas, los recursos computacionales corrían a cuenta del usuario. Esta serie de inconvenientes no casaba con la base de Prodef de acercarse a un público sin experiencia.

En este trabajo se ha diseñado una infraestructura para desplegar la herramienta Prodef como un servicio web empleando el modelo SaaS (Software como servicio). Para alcanzar este objetivo, se ha llevado a cabo un aprendizaje previo al trabajo acerca de las tecnologías y plataformas más utilizadas para este tipo de implementación. Primero, se ha contenerizado cada microservicio de la herramienta utilizando Docker. Asimismo, se ha realizado una configuración en Docker-compose capaz de poner en marcha todos los contenedores de Prodef simultáneamente. Para esta tarea ha sido necesario realizar cambios en el código tanto del back-end como del front-end. También ha sido necesario contratar un proveedor de servicios de nube como DigitalOcean. Por último, se ha desarrollado un plan de Terraform capaz de crear un servidor virtual nuevo que, automáticamente, realiza todos los pasos necesarios para poner en marcha el Docker-compose de Prodef, consiguiendo de esta forma el despliegue del servicio.

Como resultado de este Trabajo de Fin de Grado no solo se ha establecido un plan de despliegue portable a cualquier proveedor de servidores virtuales, sino que también se ha adaptado el código de Prodef para futuras tareas de este ámbito. Además, se ha dejado operativo un enlace con nombre de dominio para acceder a Prodef desde cualquier navegador.

Palabras clave: Software como servicio (SaaS), contenerización, infraestructura, despliegue, optimización, metaheurísticas

Abstract

Currently, *bio-inspired optimization algorithms* are widely used in academic settings and there is a large amount of investigation devoted to them. However, they are highly complex methods with a long learning curve, not only in understanding but also in implementation. This is why, despite being considerably useful techniques, their use does not extend much beyond the investigation environment. Prodef is a tool that arises in this context with the aim of expanding the scope of these techniques, trying to bring them closer to business environments. Prodef allows modeling problems with a block-based graphical interface, defining algorithms without writing code and generating problem instances for subsequent resolution. Initially, however, Prodef was not a tool that was easily accessible: using it required multiple dependencies to be installed manually, and a Linux terminal had to be used to launch all the microservices. To complicate matters further, computational resources were at the user's expense. This set of drawbacks did not fit with Prodef's rationale of reaching out to an inexperienced audience.

In this work, an infrastructure has been designed to deploy the Prodef tool as a web service using the SaaS (Software as a Service) model. To achieve this goal, a pre-work learning has been carried out about the most used technologies and platforms for this type of implementation. First, each microservice of the tool has been containerized using Docker. Also, a configuration has been made in Docker-compose capable of starting all the Prodef containers simultaneously. For this task it has been necessary to make changes in the code of both the back-end and front-end. It has also been necessary to contract a cloud service provider such as DigitalOcean. Finally, it has been developed a Terraform plan capable of creating a new virtual server that automatically performs all the necessary steps to launch the Docker-compose of Prodef, thus achieving the deployment of the service.

As a result of this Final Degree Project, not only a deployment plan portable to any virtual server provider has been established, but also the Prodef code has been adapted for future tasks in this area. In addition, a link with a domain name has been left operational to access Prodef from any browser.

Keywords: Software as a Service (SaaS), containerization, infrastructure, deployment, optimization, metaheuristics

Índice general

1. Introducción	1
1.1. Antecedentes y estado actual del tema	1
1.2. Actividades a realizar	2
2. Software as a Service (SaaS)	3
2.1. Docker	3
2.1.1. Imágenes	4
2.1.2. Contenedores	5
2.1.3. Funcionamiento básico de Docker	6
2.1.4. Docker Compose	8
2.2. DigitalOcean	9
2.2.1. Obtención de una servidor virtual para el despliegue	9
2.3. Terraform	10
2.3.1. Beneficios de usar Terraform	10
2.3.2. Ciclo de vida	11
2.3.3. Funcionamiento de Terraform	11
3. Prodef	13
3.1. Arquitectura	14
3.2. Funcionamiento	15
3.2.1. Modelado de un problema	15
3.2.2. Definición de una instancia	16
3.2.3. Modelado de un algoritmo	16
3.2.4. Definición de una resolución o ejecución	17
3.2.5. Compilación del problema y el algoritmo	18
3.2.6. Resolución	20
3.3. Instalación de Prodef	21
3.4. Interfaz gráfica de usuario	21
3.5. Aspectos a tener en cuenta	23
4. Contenerización de Prodef	24
4.1. Resolutor (Rust)	25
4.2. Back-end	27
4.3. Front-end	28
4.4. Prodef docker-compose	30
5. Despliegue de Prodef	32
5.1. Despliegue manual	32
5.2. Nombre de dominio	33

5.3. Despliegue con Terraform	37
5.3.1. Clave pública	37
5.3.2. Droplet	38
5.3.3. Nombre de dominio	38
6. Conclusiones y líneas futuras	41
6.1. Conclusiones	41
6.2. Líneas de trabajo futuras	42
7. Summary and Conclusions	44
7.1. Conclusions	44
7.2. Lines for future work	45
8. Presupuesto	46
A. Fichero de configuración de Terraform para definir el droplet de DigitalOcean	47

Índice de Figuras

2.1. Funcionamiento de Docker	3
2.2. Primeros resultados del listado de imágenes públicas en DockerHub	4
2.3. Estructura básica de un contenedor Docker	5
2.4. Generación de una imagen Docker de prueba	7
2.5. Creación y ejecución de un contenedor	7
2.6. Listado de contenedores activos	8
2.7. Página de selección de servicio	9
2.8. Droplet en DigitalOcean	10
2.9. Ciclo de vida de Terraform.	11
2.10. Esquema de funcionamiento de Terraform.	12
3.1. Prodef - Diseño Inicial	14
3.2. Prodef - Diseño Actual	15
3.3. Problema de la mochila definido con bloques en Prodef	16
3.4. Instancia del problema de la mochila en Prodef	17
3.5. Algoritmo GRASP simple modelado en Prodef	17
3.6. Definición de la ejecución a través de Prodef	18
3.7. Información de la ejecución en Prodef	20
3.8. Esquema de funcionamiento de Prodef	20
3.9. Prodef GUI - Página principal	22
4.1. Flujo de peticiones	24
4.2. Diagrama del funcionamiento del docker-compose de Prodef	31
5.1. Validez del nombre de dominio prodef	33
5.2. DNS de DigitalOcean	34
5.3. Configuración de servidor DNS en Namecheap	34
5.4. Información de la aplicación OAuth	35
5.5. Esquema de la creación de la infraestructura de Prodef	40

Índice de Tablas

8.1. Presupuesto por tiempo trabajado 46
8.2. Gastos extra 46

Capítulo 1

Introducción

Existe una serie de problemas reales (logística, ingeniería, etc) para los cuales no siempre existe un algoritmo de coste computacional razonable que los resuelva. Dichos problemas requieren de tareas como encontrar el camino más corto entre varios puntos, una asignación horaria de trabajadores a tareas a realizar, y muchas otras cuestiones consistentes en planificar o asignar recursos. En problemas de esta naturaleza, puede no existir una solución única, pues los mismos requisitos se pueden satisfacer de diferentes maneras. La dificultad aparece cuando tienes que maximizar o minimizar un recurso, para lo cual se debe utilizar algún tipo de estrategia (heurística o meta-heurística) que nos lleve a una solución óptima o al menos lo más cercana posible a la óptima.

Los algoritmos metaheurísticos son algoritmos aproximados de optimización y búsqueda para resolver un tipo de problema computacional general. Son procedimientos iterativos que combinan inteligentemente diversas técnicas para explorar el espacio de soluciones. Estas estrategias nos permiten encontrar soluciones satisfactorias en un tiempo bastante razonable, lo que los convierte en una herramienta extremadamente útil para tareas de organización y asignación. El problema reside en que de manera cotidiana, las entidades ajenas al campo de la investigación y la programación no tienen los conocimientos necesarios para aplicar este tipo de algoritmos. Por este motivo, habitualmente estos procesos de optimización terminan realizándose de forma manual, empleando muchos recursos (como tiempo o personas) y sin tan siquiera tener la certeza de estar obteniendo soluciones “aceptables” (u óptimas, según el caso).

1.1. Antecedentes y estado actual del tema

Prodef es una herramienta que nace con el objetivo de extender el uso de estas técnicas meta-heurísticas fuera del ámbito de la investigación convencional, tratando incluso de acercarlas a pequeñas y medianas empresas. Para ello, proporciona una interfaz gráfica para el modelado del problema lo cual supone un salto cualitativo en lo que respecta a sencillez de uso, dejando atrás complejas implementaciones a nivel de código o configuraciones a bajo nivel. La herramienta permite, a través de una interfaz web, modelar un problema, especificar una o más instancias del mismo y resolverlo directamente sin necesidad de escribir ni una línea de código. De esta forma, los usuarios pueden aplicar técnicas meta-heurísticas para la resolución de sus problemas obteniendo, en la medida de lo posible, soluciones razonables para sus problemas. Todo esto aislando al usuario de la implementación interna y de conceptos relacionados con las metaheurísticas.

El propósito último de Prodef es proporcionar una herramienta capaz de proponer soluciones para problemas de optimización mediante un meta-modelo sencillo que una persona familiarizada con el problema, pero no necesariamente con la programación o las metaheurísticas, pueda usar sin ayuda de un experto. Prodef se basa en un meta-modelo que permite abstraer las características esenciales de un problema de optimización combinatoria. A partir de este modelo, se genera una traducción directa a una herramienta externa de optimización basada en metaheurísticas bio-inspiradas.

En su estado actual, es posible obtener soluciones para problemas de optimización usando Prodef y alguno de sus resolutores, como el de jMetal [9]. Sin embargo, el proyecto sigue circunscrito a una serie de repositorios de código, y por tanto, se requiere de un usuario experto para poder ejecutarlo. Hasta el momento no existe una infraestructura que permita a un usuario interactuar (o ejecutar) con Prodef de forma sencilla. Por este motivo, sería interesante investigar las diferentes alternativas que tiene la herramienta para poder llegar a su público objetivo. Se hablará en profundidad sobre Prodef en el Capítulo 3.

1.2. Actividades a realizar

Las tareas que se engloban como parte del desarrollo de este Trabajo de Fin de Grado son las siguientes:

1. **Contenerización de los resolutores de Prodef.** El objetivo es conseguir que los resolutores se puedan desplegar fácilmente en cualquier máquina Linux. En el caso del resolutor de jMetal, por ejemplo, consistiría en crear una imagen Docker con todas sus dependencias, incluyendo Java y el propio jMetal.
2. **Contenerización del front-end de Prodef.** Prodef dispondrá de uno o más frontends web con los que poder interactuar de forma gráfica, ya sea para modelar el problema, modelar el algoritmo o para visualizar las soluciones propuestas. Se deberá contenerizar también estas aplicaciones web.
3. **Diseño de la arquitectura de despliegue.** Una vez que todos los componentes de la herramienta que se necesiten desplegar estén contenerizados se podría empezar a diseñar la propia arquitectura de despliegue.
4. **Despliegue de la herramienta.** El objetivo es poder acceder a la herramienta por Internet simplemente conectándose a un sitio web.
5. **Generación de documentación.** Elaboración del material de soporte para el uso de la herramienta así como la memoria del TFG y demás documentación requerida durante la realización de la asignatura de TFG.

Cabe mencionar que el alumno no tiene conocimiento previo en las herramientas y conceptos a utilizar en este Trabajo de Fin de Grado (contenerización, Docker, SaaS, despliegue de Software). Debido a esto, una parte importante del Trabajo de Fin de Grado es aprender y utilizar estas tecnologías.

Capítulo 2

Software as a Service (SaaS)

A día de hoy, el modelo de negocio que predomina en la industria es *Everything as a Service (XaaS)* [31]. En el caso de la industria del software tenemos *Software as a Service (SaaS)* [3]. Este modelo consiste en proporcionar la herramienta como un servicio disponible en Internet, de forma que el usuario no tenga que instalar ningún *software* ni preocuparse de aspectos técnicos para poder hacer uso del servicio deseado. Con SaaS es posible ofrecer a los usuarios finales aplicaciones relativamente complejas de una forma casi transparente, sin necesidad de adquirir, instalar, actualizar o mantener ningún tipo de hardware, middleware o software. De esta forma, aplicaciones empresariales sofisticadas quedan al alcance de organizaciones que no cuentan con grandes recursos para adquirir infraestructuras ni implementar soluciones a medida. En este capítulo se presentan las tecnologías y plataformas de SaaS que se van a utilizar: Docker, DigitalOcean y Terraform.

2.1. Docker

Docker [18] es una herramienta de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores aislados y reproducibles autosuficientemente. La idea detrás de Docker es crear contenedores ligeros y portables para que las aplicaciones software puedan ejecutarse en cualquier máquina con Docker instalado, independientemente del sistema operativo local, facilitando de esta forma los despliegues. Los dos elementos más básicos de Docker son las imágenes y los contenedores.

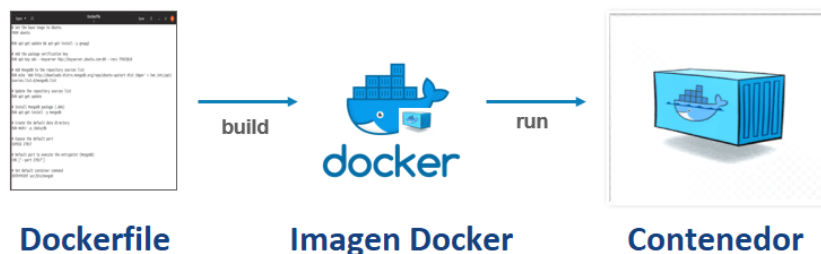


Figura 2.1: Funcionamiento de Docker

2.1.1. Imágenes

Una imagen es una especie de plantilla para crear contenedores, una captura del estado de un contenedor. Las imágenes son creadas a partir de un fichero de configuración llamado `dockerfile` (Figura 3.8). Un `dockerfile` es un script que contiene una colección de comandos e instrucciones que serán ejecutados automáticamente en el entorno Docker para construir una nueva imagen. Las imágenes nunca cambian.

[DockerHub](#) es un servicio proporcionado por *Docker* para compartir imágenes. Además, cuenta con un sitio web que resulta de utilidad para explorar las imágenes que se han publicado, tal y como se puede apreciar en la Figura 2.2. Hay muchas imágenes oficiales con elementos básicos como *Java*, *Ubuntu*, *Nginx*, etc, que se pueden descargar y utilizar.

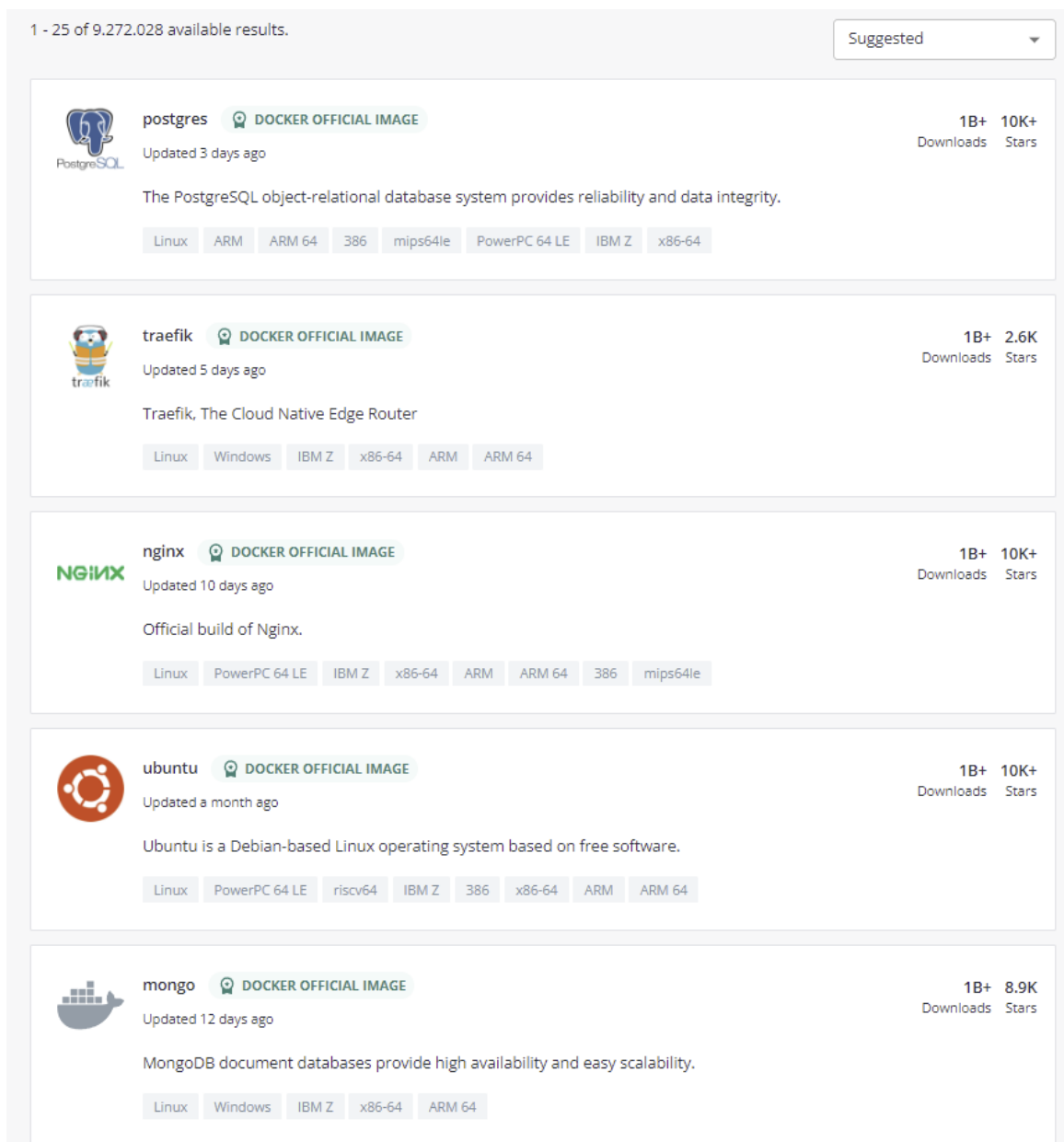


Figura 2.2: Primeros resultados del listado de imágenes públicas en DockerHub

Volviendo a la escritura de un `dockerfile`, el lenguaje de Docker cuenta con una breve lista de *keywords* [23]:

- **FROM:** especifica la imagen base de la que se quiere partir.
- **RUN:** ejecuta un comando específico de Linux en la imagen.
- **WORKDIR:** establece el directorio de trabajo dentro de la imagen que se está construyendo.
- **COPY:** copia el contenido de un directorio en la ubicación especificada.
- **USER:** cambia el usuario actual de la imagen (por defecto es *root*). Dicho cambio se mantendrá incluso en la creación de los contenedores.
- **EXPOSE:** cuando se cree el contenedor, expone un puerto de este hacia la máquina local.
- **ENTRYPOINT** o **CMD:** establece un comando de Linux que se ejecutará una vez se cree el contenedor, no durante la fase de construcción.

2.1.2. Contenedores

Los contenedores son instancias en ejecución de una imagen cuya utilidad es ejecutar nuestra aplicación. Para que un contenedor pueda realizar esta tarea necesitaría: un sistema operativo sobre el que trabajar, tener instaladas las dependencias que utilice el software contenerizado y, por último, el código de la aplicación en sí.

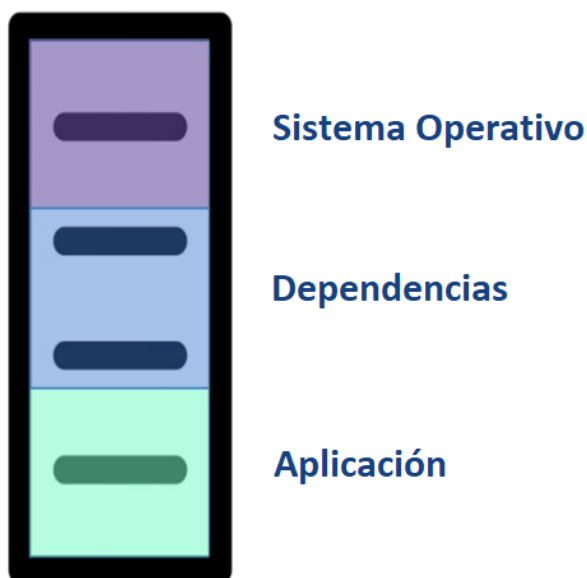


Figura 2.3: Estructura básica de un contenedor Docker

A partir de una única imagen se pueden ejecutar múltiples contenedores. Esto es una buena manera de tener copias de la aplicación ejecutándose en varios contenedores, para luego, a través de balanceadores de carga, distribuir los accesos al software y ofrecer servicios con más garantías y menos sobrecarga de peticiones.

2.1.3. Funcionamiento básico de Docker

Con el fin de presentar el funcionamiento de Docker, se ha realizado un primer ejemplo de contenerización consistente en poner en marcha una aplicación web muy sencilla siguiendo un tutorial [22]. El primer paso ha sido instalar Docker y aprender los comandos simples (construir una imagen, visualizar el listado de imágenes, crear un contenedor a partir de una imagen, visualizar el listado de contenedores activos, detener la ejecución de un contenedor, etc). Todos los ficheros escritos durante este tutorial se han almacenado en un directorio creado únicamente para esta tarea.

La aplicación ha consistido en el siguiente fichero html:

```
1 <h4>Aplicación web contenerizada</h4>
2
3 <script>
4   confirm('¿estoy en un contenedor?');
5 </script>
```

Una vez creada la aplicación, se ha procedido a escribir el `dockerfile`, (que va situado en el directorio raíz, junto con el fichero html), en el cual se especifican las dependencias necesarias para desplegar la web, que en este caso son un sistema operativo con núcleo Linux y un servidor web como *Nginx*.

Es una práctica común utilizar una imagen ya existente como base, ya que es ineficiente instalar, comando por comando, un servidor web teniendo la posibilidad de incluir una imagen que ya tenga las dependencias de la herramienta y que ya traiga el sistema operativo. Volviendo a la Figura 2.2, es observable que existen imágenes oficiales de *Nginx*, por lo que se ha decidido usar una de ellas. Concretamente se va a utilizar "nginx:1.19.0-alpine":

```
1 FROM nginx:1.19.0-alpine
```

A continuación, los ficheros de código fuente se deben guardar dentro del contenedor, ya que por ahora solo se tiene el sistema operativo y *Nginx* instalados. Es conveniente leer la documentación del servidor web que se utilice [29]. En este caso, se necesita conocer dónde se deben guardar los archivos para un servidor de ficheros estáticos. Para *Nginx*, se debe ubicar la aplicación web en la ruta `/usr/share/nginx/html`, por lo que lo siguiente que debe hacer el `dockerfile` es copiar el contenido del directorio en dicha ubicación. Este sería el `dockerfile` terminado:

```
1 FROM nginx:1.19.0-alpine
2 COPY . /usr/share/nginx/html
```

Ahora viene la parte de los comandos. Lo primero es generar una imagen a partir de nuestro `dockerfile`. Para ello se utiliza la siguiente instrucción:


```
$ docker build -t docker_test_angel .
```

Con esto, se crea una nueva imagen Docker llamada `docker_test_angel`. También se le está indicando que los ficheros necesarios se encuentran en el directorio actual.

```
angel@Angred:~/Prodef_SAAS/docker_test$ docker build -t docker_test_angel .
Sending build context to Docker daemon 3.072kB
Step 1/2 : FROM nginx:1.19.0-alpine
--> 7d0cdcc60a96
Step 2/2 : COPY . /usr/share/nginx/html
--> 75a9c9e74bca
Successfully built 75a9c9e74bca
Successfully tagged docker_test_angel:latest
angel@Angred:~/Prodef_SAAS/docker_test$
```

Figura 2.4: Generación de una imagen Docker de prueba

Para crear un contenedor a partir de esta nueva imagen, se emplea el siguiente comando:

```
$ docker run -p 80:80 docker_test_angel
```

Con la opción `-p 80:80` se está indicando que hay una máquina interna exponiendo su puerto 80 hacia el puerto 80 de la máquina local, ya que es en esta desde la cual se va a acceder al navegador. Particularmente es el 80 porque es el puerto por defecto de *Nginx*.

```
angel@Angred:~/Prodef_SAAS/docker_test$ docker run -p 80:80 docker_test_angel
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
```

Figura 2.5: Creación y ejecución de un contenedor

Ya se tendría el contenedor creado. Tras esto, ya se podría visualizar la página web en el navegador. Se puede verificar que el contenedor está activo ejecutando el siguiente comando, el cual lista los contenedores activos:

```
$ docker ps
```

Este comando proporciona información útil sobre cada contenedor activo, por ejemplo un identificador que servirá para referirse a este en otros comandos. Añadiendo la opción `-a` se visualizarán también los contenedores inactivos.

Por último, para eliminar el contenedor (que no es lo mismo que ponerlo en estado inactivo), se utiliza el siguiente comando:

```

angel@Angred:~/Prodef_SAAS/docker_test$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS
eec833dddfb3   docker_test_angel  "/docker-entrypoint..."  5 minutes ago  Up 4 minutes  0.0.0.0:80->80/tcp, :::80->80/tcp
angel@Angred:~/Prodef_SAAS/docker_test$

```

Figura 2.6: Listado de contenedores activos

```
$ docker rm [CONTAINER ID]
```

2.1.4. Docker Compose

Las aplicaciones basadas en microservicios se prestan a usar múltiples contenedores (cada uno con un servicio) que se relacionan entre ellos. Por ejemplo: uno puede contener la base de datos *SQL*, otro un sistema de mensajería como *WildFly* y un servidor web como *Nginx*. Docker Compose [20] es una herramienta de Docker que sirve para definir y ejecutar aplicaciones **multicontenedor**.

Para utilizarlo, se emplea un fichero de texto en formato `.yml` con las mismas propiedades que se indicarían con el comando `docker run` individualmente, así como las dependencias entre los contenedores. Una vez definidos los múltiples contenedores del servicio en el archivo, *Docker compose* permite iniciar con un único comando, `docker-compose up`, todos los contenedores en el orden especificado según sus dependencias.

El archivo descriptor puede servir no solo como forma de iniciar los contenedores en un entorno de desarrollo sino como de documentación de la aplicación en la que se muestra qué contenedores, imágenes, opciones, enlaces y demás propiedades tienen. A continuación, se muestra un ejemplo de un fichero `docker-compose.yml` que define como servicio la aplicación contenerizada de la sección anterior:

```

1 version: "3.0"
2 services:
3   test:
4     build:
5       context: docker_test/
6     image: docker_test_angel
7     ports:
8       - "80:80"

```

En este caso, solo está definido un servicio llamado *test*. El apartado *build* sirve para especificar la ruta del Dockerfile que crea la imagen en caso de que se quiera reconstruir antes de ejecutarla, *image* indica la imagen asociada al servicio y *ports* funciona igual que la opción `-p 80:80`, es decir, para exponer el puerto 80 del contenedor al puerto 80 local.

2.2. DigitalOcean

DigitalOcean [16] es un proveedor de servidores virtuales privados. Esta plataforma maneja el concepto de *droplet* (que traducido es “gotita”) para designar a cada una de las máquinas virtuales, las cuales ofrecen en alquiler. Son privadas porque DigitalOcean no interviene en su instalación y manejo, limitándose únicamente a ofrecer imágenes de los principales sistemas operativos.

2.2.1. Obtención de una servidor virtual para el despliegue

Obtener un *droplet* en esta plataforma es sencillo, pues solo requiere registrarse en la página web y autenticarse con un método de pago, que se confirmará realizando un pago de 1 USD (95 céntimos). Una vez realizado este inicio de sesión, la página ofrece directamente todos sus servicios como se muestra en la Figura 2.7.

Es necesario realizar un formulario en el que se especifican las características del servidor virtual que se quiere contratar. En este caso se ha optado por elegir una máquina Ubuntu lo más simple posible en cuanto a recursos (5,76€/mes) ya que el objetivo es una prueba de concepto. Una de las ventajas de *DigitalOcean* es que se puede cambiar a un plan mejor en cualquier momento y, al facturar por horas, en caso de aumento de recursos el precio solo aumentaría durante las horas que esté activo dicho plan. Se ha indicado que la ubicación del servidor deseada es Frankfurt (Alemania) debido a que era la opción con mayor cercanía a las Islas Canarias. Existía otra alternativa en Reino Unido pero se descartó para evitar posibles problemas de protección de datos al no ser este país de la Unión Europea. En este formulario de creación de máquina virtual, se ofrece la posibilidad de establecer una *ssh-key* para acceder a esta mediante SSH, lo cual ha sido de interés y por tanto se ha aprovechado.

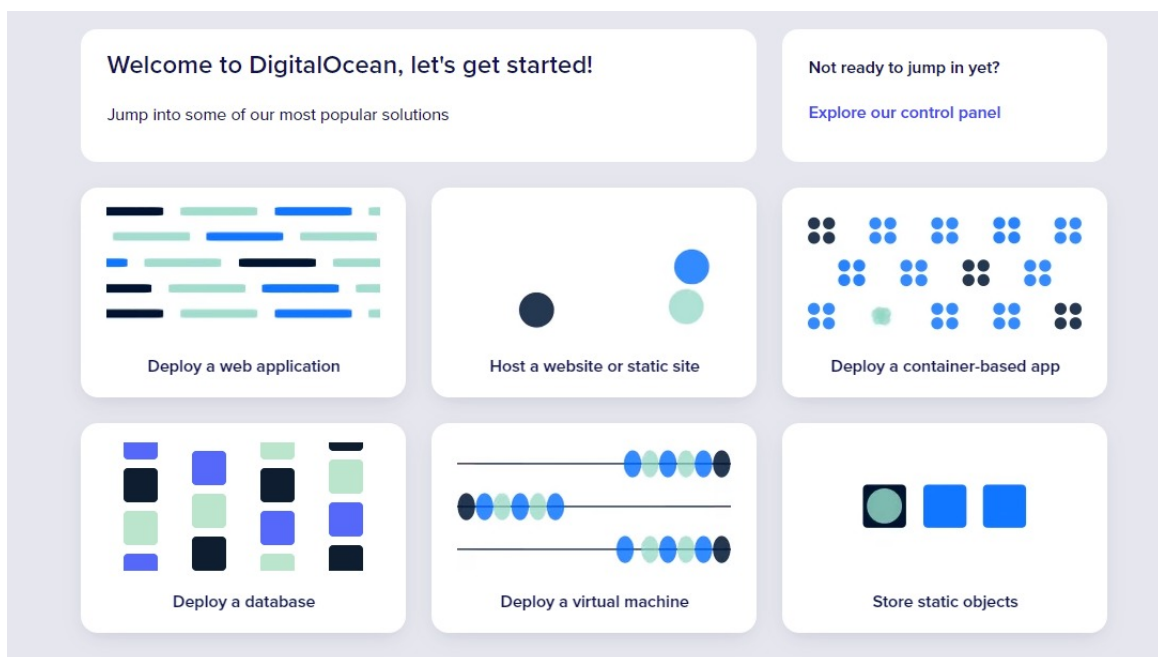


Figura 2.7: Página de selección de servicio

Una vez rellenado el formulario, el *droplet* ya está activo y disponible para conectarse mediante ssh y comenzar con el trabajo. La Figura 2.9 muestra la vista que se tiene de la máquina virtual desde *DigitalOcean*.



Figura 2.8: Droplet en DigitalOcean

2.3. Terraform

Terraform [13] [1] es una tecnología que permite declarar toda una infraestructura a través de código. Permite a los usuarios definir y configurar la infraestructura de un servidor mediante código de un lenguaje de alto nivel, generando un plan de ejecución para desplegarla. Terraform permite incluso administrar nombres de dominio.

2.3.1. Beneficios de usar Terraform

- Terraform se puede conectar a la API de diferentes proveedores, por ejemplo *Google*, *Amazon* el propio *DigitalOcean*, etc.
- Proporciona una infraestructura inmutable donde la configuración puede cambiarse sin problemas.
- Utiliza un lenguaje fácil de entender, HCL (*HashiCorp Configuration Language*).
- Se puede cambiar de proveedor de forma sencilla.
- Admite arquitectura de solo cliente, por lo que no es necesario administrar la configuración adicional en un servidor.

2.3.2. Ciclo de vida

El ciclo de vida de Terraform consiste en cuatro fases:



Figura 2.9: Ciclo de vida de Terraform.

1. `terraform init` inicializa el directorio de trabajo, que consiste en el conjunto de todos los ficheros de configuración.
2. `terraform plan` es usado para crear un plan de ejecución basado en el contenido de todos los archivos de configuración que se encuentren en el directorio de trabajo.
3. `terraform apply` aplica los cambios en la infraestructura tal y como están definidos en el plan.
4. `terraform destroy` se usa para eliminar todos los recursos de infraestructura antiguos.

2.3.3. Funcionamiento de Terraform

Terraform tiene dos componentes principales que conforman su arquitectura:

- **Núcleo de Terraform.** Este utiliza dos fuentes de entrada para hacer su trabajo:
 1. La primera fuente de entrada son los ficheros de configuración de Terraform que el usuario configura (ficheros de tipo `.tf`). En este apartado se define qué recursos necesitan ser provisionados.
 2. La segunda fuente de entrada es un fichero de tipo *Terraform State* (`.tfstate`) en el que Terraform mantiene actualizado cuál es el estado de la configuración actual de la infraestructura.
- **Proveedores.** El segundo componente son proveedores para servicios específicos. Estos pueden ser proveedores de nube como *Azure* o cualquier plataforma de IaaS (*Infrastructure as a Service*) [31].

En la Figura 2.10 se pueden ver los conceptos explicados (referido al funcionamiento de Terraform) de manera esquematizada.

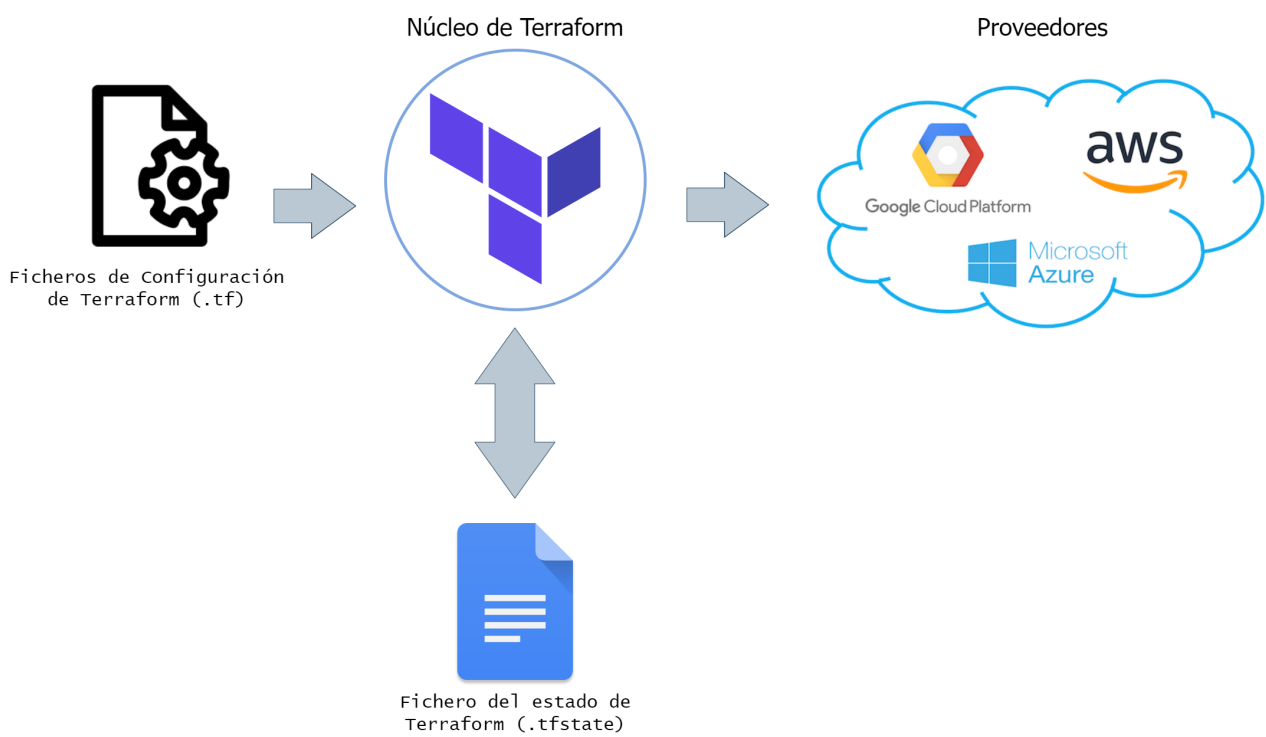


Figura 2.10: Esquema de funcionamiento de Terraform.

Capítulo 3

Prodef

Tal y como ya se ha mencionado, Prodef es una herramienta que nace con el objetivo de extender el uso de las meta-heurísticas fuera del ámbito de la investigación convencional, tratando incluso de acercarlas a pequeñas y medianas empresas. Para ello, se centra en ofrecer interfaces gráficas para el modelado y la definición de los problemas, así como un sistema interno de resolución que permita a los usuarios abstraerse totalmente de las técnicas que se utilizarán durante el proceso de optimización.

La implementación actual de Prodef es el resultado de varios Trabajos de Fin de Grado:

- **Prodef: Meta-modelado de problemas de optimización combinatoria** (2020). *Andrés Calimero Garcia Pérez* [11]. En este trabajo se implementó la estructura inicial de Prodef, una herramienta que permite la definición de problemas de optimización combinatoria a un nivel de descripción abstracto y su posterior resolución usando librerías y frameworks externos, como jMetal [9]. La Figura 3.1 permite ver la arquitectura inicial de Prodef. En el esquema, el símbolo (*) representa que el módulo proporciona una clase abstracta o interfaz y se debe utilizar alguna implementación; en este caso, al utilizar jMetal, se implementó un módulo de compilación `prodef-compiler-java` y uno de resolución `prodef-solver-jmetal`.
- **Prodef-GUI: Interfaz gráfica para el modelado de problemas** (2021). *Daniel González Expósito* [12]. En este trabajo se desarrolló la interfaz gráfica de Prodef: Prodef-GUI. Gracias a esta interfaz web los usuarios pueden modelar sus problemas de optimización (variables, objetivos, restricciones, así como las instancias de entrada) conectando visualmente bloques.
- **Prodef: Diseño, implementación y experimentación con nuevos resolutores** (2022). *Miguel Ángel Ordóñez Morales* [26]. En este trabajo se desarrolló un resolutor en C++ utilizando la herramienta METCO [5] con la finalidad de añadir otra manera (además del resolutor basado en jMetal ya previamente incorporado) de resolver los problemas en Prodef.
- **Prodef-Algorithm: Interfaz para el modelado de meta-heurísticas** (2022). *Daniel del Castillo de la Rosa* [6]. En este trabajo se han implementado los cambios necesarios para que Prodef permita al usuario definir sus propios algoritmos de forma sencilla, además de un nuevo resolutor en el lenguaje Rust. Este último se diferencia de los demás en que el código que resuelve el problema no es un método estático y predefinido, sino que se genera en tiempo de ejecución según el algoritmo definido en la interfaz.

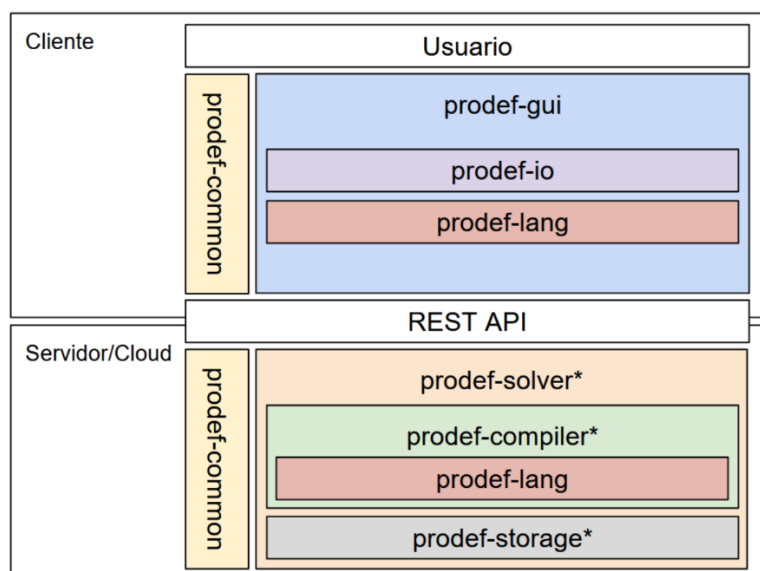


Figura 3.1: Prodef - Diseño Inicial

3.1. Arquitectura

Actualmente, Prodef es un conjunto de módulos que interactúan entre sí. Se ha realizado un estudio de su estructura actual (Figura 3.2), tratando de comprender la funcionalidad de cada uno de sus componentes. Este análisis de la arquitectura ha sido de utilidad debido a que, para poder contenerizar las diferentes partes de la herramienta, es necesario conocer qué elementos son necesarios en cada una de ellas:

- **prodef-common**: un pequeño módulo que proporciona interfaces y los tipos de datos comunes que utiliza Prodef.
- **prodef-io**: implementa distintas herramientas para deserializar y validar problemas, algoritmos y soluciones en formato JSON.
- **prodef-lang**: contiene la definición del lenguaje de dominio específico de Prodef.
- **prodef-compiler**: define un compilador de ProdefLang a otro lenguaje. Se ha implementado de forma genérica de manera que se pueda transpilar a distintos lenguajes.
- **prodef-solver**: contiene la definición de un resolutor, así como un servidor que puede ser usado con cualquier implementación de un resolutor.
- **prodef-gui-backend**: sirve de intermediario entre el resolutor y el *front-end* y permite almacenar en una base de datos *MongoDB* [21] problemas, algoritmos, instancias y ejecuciones.
- **prodef-secrets**: consta de un fichero que almacena valores importantes secretos para hacer funcionar el *back-end*.
- **prodef-gui-frontend**: implementa la interfaz gráfica de usuario utilizando la librería *React*. Permite modelar problemas y algoritmos gráficamente.

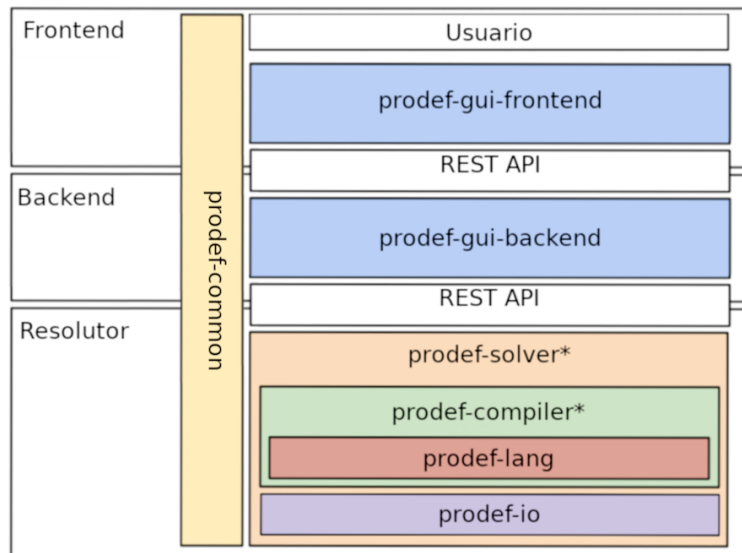


Figura 3.2: Prodef - Diseño Actual

3.2. Funcionamiento

Esta sección está dedicada al estudio en profundidad del funcionamiento de Prodef. Concretamente se explicará qué pasa internamente desde que se define un problema en la interfaz hasta que es resuelto.

3.2.1. Modelado de un problema

Se debe modelar un problema gráficamente usando una interfaz gráfica basada en Blockly [10]. Cabe destacar que entender esta interfaz requiere conocimientos en notación matemática. En este apartado se utilizará el problema de la mochila a modo de ejemplo, que se puede ver definido utilizando bloques en la Figura 3.3. La definición de un problema consta de cuatro apartados:

- **Datos de entrada.** Define los parámetros que se deben pasar al problema. En el caso del problema la mochila, los datos de entrada son el valor de *maxWeight* (el peso de la mochila) y una tabla de *Items* con una columna para el valor y otra para el peso de los objetos.
- **Variables.** Establece las variables que conforman una solución. Estas pueden ser listas, listas permutadas, matrices o una cifra única. Exceptuando la lista permutada, todos los tipos de variable aceptan tanto números enteros como números reales y permiten establecer un rango de valores. En nuestro ejemplo se puede ver que la solución es una lista de enteros, en la que cada posición corresponde a un objeto y el valor de cada posición indica si el objeto pertenece o no a la solución.
- **Objetivos:** La función objetivo del problema, es decir, cómo se deben utilizar los parámetros y variables para alcanzar el valor objetivo.
- **Restricciones:** Definen si una solución es o no factible. Una solución solo es factible si cumple todas las restricciones. Por ejemplo, en el problema de la mochila, existe

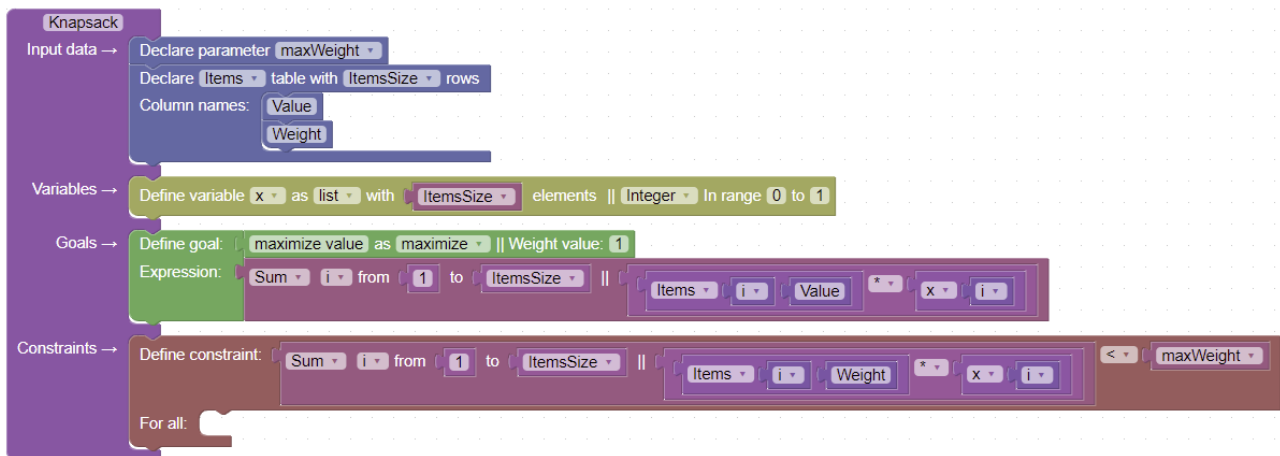


Figura 3.3: Problema de la mochila definido con bloques en Prodef

la restricción de que la suma del peso de todos los objetos no debe superar el valor de *maxWeight*.

3.2.2. Definición de una instancia

Una vez modelado el problema, es necesario crear una instancia de este. Tanto el problema como las instancias se guardan por separado ya que una problema puede tener múltiples instancias. Prodef permite definir las desde la propia interfaz, pero también es posible subir un fichero en formato .csv con los valores requeridos. La Figura 3.4 muestra una instancia de ejemplo para el problema de la mochila.

3.2.3. Modelado de un algoritmo

En la última actualización de Prodef [6] se implementó la funcionalidad de modelar algoritmos. Esto se hace en una interfaz basada en Blockly al igual que con la definición de loss problemas. Cabe destacar que no es necesario que el usuario defina su propio algoritmo para resolver un problema; Prodef cuenta con algoritmos públicos ya que no hay que olvidar que el objetivo de la herramienta es llevar las técnicas meta-heurísticas fuera del ámbito de la investigación convencional. El modelado consiste en añadir componentes en el orden deseado. Prodef cuenta con tres tipos de componentes:

- **Seguro:** no permiten que en el flujo principal del algoritmo existan soluciones no factibles (bloques amarillos).
- **No seguro:** permiten soluciones no factibles (bloques rojos).
- **Válido para ambos casos:** pueden usarse junto con los dos otros tipos de componente (bloques naranjas).

En la Figura 3.5 se puede ver una versión simple del algoritmo GRASP (*Greedy Randomized Adaptive Search Procedure*) [2]. Es cierto que un algoritmo de esta naturaleza requiere una serie de parámetros, como por ejemplo la forma concreta de generar soluciones o el número de veces que debe repetirse el bucle; dichos valores se rellenan cuando se vaya a ejecutar el algoritmo con un problema y una instancia concretos.

Instances Select Instance ▾

Name: example instance

Parameters

Name	Value
maxWeight	27

Data tables

Items

Value	Weight
3	8
4	1
8	3
1	4
4	6
9	2
3	8
7	7
1	4
2	1
5	7

[Download instance csv](#)

Figura 3.4: Instancia del problema de la mochila en Prodef

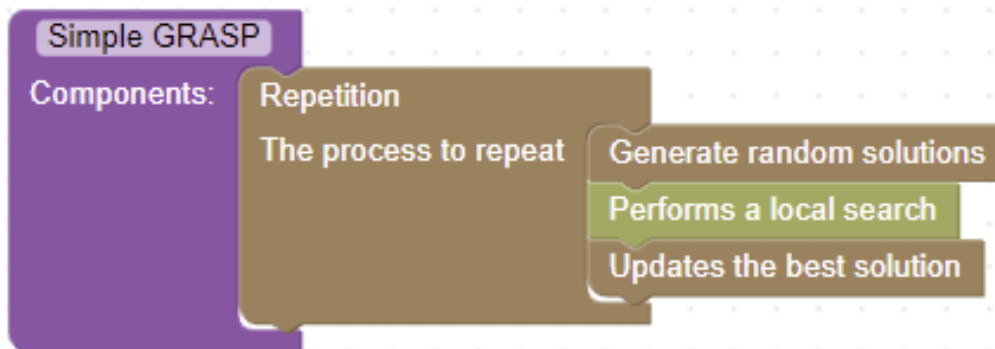


Figura 3.5: Algoritmo GRASP simple modelado en Prodef

3.2.4. Definición de una resolución o ejecución

Una vez definidos el problema, una instancia y un algoritmo, se puede mandar una petición de ejecución a través del *back-end*. Para ello, es preciso dar valor a los parámetros que utiliza el algoritmo. En la Figura 3.6 se muestra la vista que tendríamos del formulario del algoritmo GRASP que está definido en la Figura 3.5.

Executions

▾

▾

FixedRepetition

The number of times to repeat the process:

The process to repeat:

BasicGenerator

The specific generator to use: ▾

The lower bound:

The upper bound:

The number of solutions to generate:

LocalSearch

The specific neighborhood to use: ▾

The change to apply:

UpdateBestSolution

with name

Figura 3.6: Definición de la ejecución a través de Prodef

3.2.5. Compilación del problema y el algoritmo

Aunque tanto el problema como el algoritmo se definen usando la interfaz gráfica, al mandar la ejecución se guardan y se envían al resolutor en un formato JSON concreto que contiene ciertos campos con expresiones en ProdefLang. Cuando el problema y el algoritmo llegan al resolutor, primero deben compilarse. Esto hace que:

1. Las expresiones del problema escritas en ProdefLang se transformen en expresiones adecuadas para el lenguaje del resolutor.
2. La definición de la secuencia de componentes del algoritmo se transpilen a un algoritmo unificado escrito en Rust.

La razón de que en este trabajo nos centremos en el resolutor desarrollado en Rust es debido a que, como ya se comentó, por ahora es el único resolutor de Prodef que soporta el modelado de algoritmos. A continuación se va a mostrar un ejemplo de transpilación utilizando el componente de repetición del algoritmo GRASP definido en la Figura 3.5.

Definición del componente:

```
1 {
2   "name": "FixedRepetition",
3   "title": "Repetition",
4   "description": "Repeats a process for a fixed number of times",
5   "safety": "both",
6   "dependencies": [
7     {
8       "type": "integer",
9       "value": 1,
10      "title": "The number of times to repeat the process"
11    },
12    {
13      "type": "algorithm",
14      "algorithm": {
15        "components": []
16      },
17      "title": "The process to repeat"
18    }
19  ]
20 }
```

A continuación, se puede ver el código en Rust que implementa este componente tras la compilación:

```
1 pub struct FixedRepetition<P: Problem<S>, S> {
2     times: i64,
3     algorithm: Algorithm<P, S>,
4 }
5 impl<P: Problem<S>, S: Clone> MainComponent<P, S> for FixedRepetition<P, S> {
6     fn apply(&mut self, problem: &mut P, mut solutions: Vec<S>) -> Vec<S> {
7         for _ in 0..self.times {
8             solutions = self.algorithm.run_components(problem, solutions);
9         }
10        solutions
11    }
12 }
13
14 impl<P: Problem<S>, S> FixedRepetition<P, S> {
15     pub fn new(times: i64, algorithm: Algorithm<P, S>) -> Self {
16         Self { times, algorithm }
17     }
18 }
```

3.2.6. Resolución

Una vez el problema y el algoritmo ya han sido compilados, el resolutor puede utilizar estas expresiones para realizar la ejecución y obtener el resultado del problema. Cuando se ha obtenido el resultado, este se guarda en la base de datos gracias al *back-end* y se puede mostrar al usuario en la interfaz gráfica.

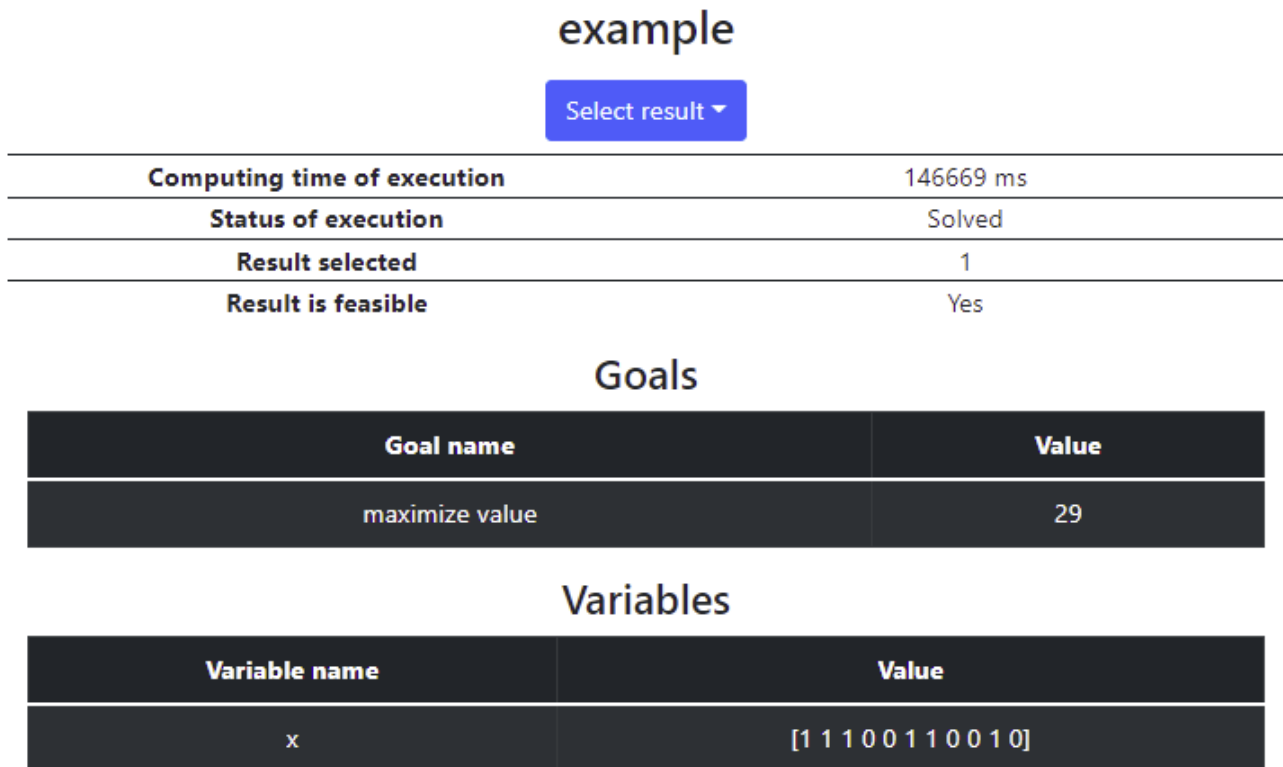


Figura 3.7: Información de la ejecución en Prodef

En la Figura 3.7 se puede ver que, al acceder a la resolución, es posible conocer cierta información como puede ser el tiempo de ejecución, el estado de la resolución (permite saber si se ha calculado correctamente o ha habido algún problema interno), si el resultado alcanzado es factible, etc. A continuación, se muestra la solución a la que se ha llegado. En este caso, al ser el problema de la mochila, se enseña cuál ha sido la suma total del valor de los objetos que conforman la solución y, por supuesto, qué objetos forman parte de la solución obtenida. La Figura 3.8 representa un resumen de todo el proceso explicado.

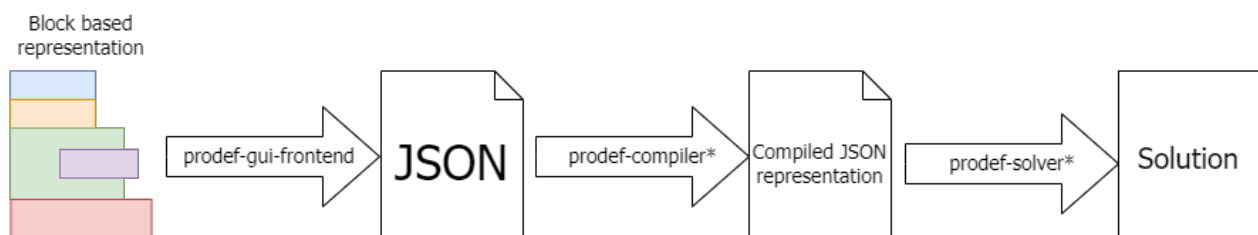


Figura 3.8: Esquema de funcionamiento de Prodef

3.3. Instalación de Prodef

Ser capaz de ejecutar Prodef de manera local es crucial si se quiere proceder a desplegar la aplicación. Entre otras cosas, permite conocer de cerca todas las dependencias a bajo nivel. Por ello, se ha procedido a instalar la primera versión de Prodef (que cuenta con un resolutor de Jmetal). Estos son los pasos seguidos para instalar y ejecutar Prodef:

1. Clonar el meta-repositorio <https://github.com/ULL-prodef/prodef.git>. Este utiliza la herramienta *meta* [30], que sirve para gestionar librerías y sistemas multi-proyecto. En el repositorio encontramos un fichero `.meta`, con la misma estructura que un JSON, que tiene definidos todos los repositorios de Prodef con el fin de clonarlos a la vez y manejarlos de manera centralizada desde ese directorio de trabajo.
2. Instalar las dependencias del meta-repositorio con `npm install`.
3. Ejecutar el script `npm run update` para clonar todos los repositorios especificados en el fichero `.meta`.
4. Crear un fichero `.npmrc` con un token personal de GitHub y copiarlo a todos los repositorios que se han clonado.
5. Ejecutar los scripts para instalar, compilar y enlazar:

```
npm run install:all
npm run build:all
npm run link:all
```

6. Desde el directorio `prodef-solver-jmetal`, ejecutar el comando `npm run start`.

Una vez el servidor está activo, se puede resolver problemas utilizando la herramienta `curl`, realizando una petición *POST* para enviar un problema y *GET* para obtener el resultado.

3.4. Interfaz gráfica de usuario

Poner a funcionar la interfaz gráfica de Prodef localmente es un proceso que también tiene su complejidad y, de hecho, se emplean dos de los tres servidores que se necesita poner en marcha para tener la herramienta funcionando por completo (*back-end* y *front-end*). A continuación, se describen los pasos para este proceso:

1. Clonar el repositorio `prodef-secrets` junto con los demás. Dicho repositorio contiene un fichero llamado `prodef-backend.json` que declara un objeto con las siguientes propiedades:
 - **mongoURI**: identificador de recursos uniforme de la base de datos MongoDB que permite la conexión con esta.
 - **clientID**: identificador de la aplicación OAuth que utiliza Prodef para la autenticación de los usuarios a través de GitHub.
 - **clientSecret**: clave privada de la aplicación OAuth de GitHub.

- **serverPort**: puerto en el que se va a alojar el servidor. 5000 actualmente.
2. Ejecutar el comando `npm run start` en el directorio `prodef-gui-backend`. Esto pone en marcha el servidor del backend. Entre otras cosas, permite enviar al *frontend* información como problemas, algoritmos, instancias, ejecuciones, etc.
 3. Ejecutar el comando `npm run start` en el directorio `prodef-gui-frontend`, que inicia un servidor de desarrollo con una aplicación de *React* y permite acceder a la GUI de Prodef desde el navegador (Figura 3.9).

Actualmente la aplicación web permite:

- Registrarse e iniciar sesión. Esto se lleva a cabo a través de GitHub gracias a una aplicación OAuth.
- Crear, modificar y guardar problemas.
- Cargar problemas desde la base de datos. Un usuario puede cargar un problema creado por él o un problema público.
- Crear, modificar y guardar algoritmos.
- Cargar algoritmos desde la base de datos. Igual que con los problemas, un usuario puede cargar un algoritmo creado por él o un algoritmo público
- Generar instancias de problemas y almacenarlas en la base de datos.
- Calcular soluciones, eligiendo el algoritmo a utilizar y sus parámetros requeridos.

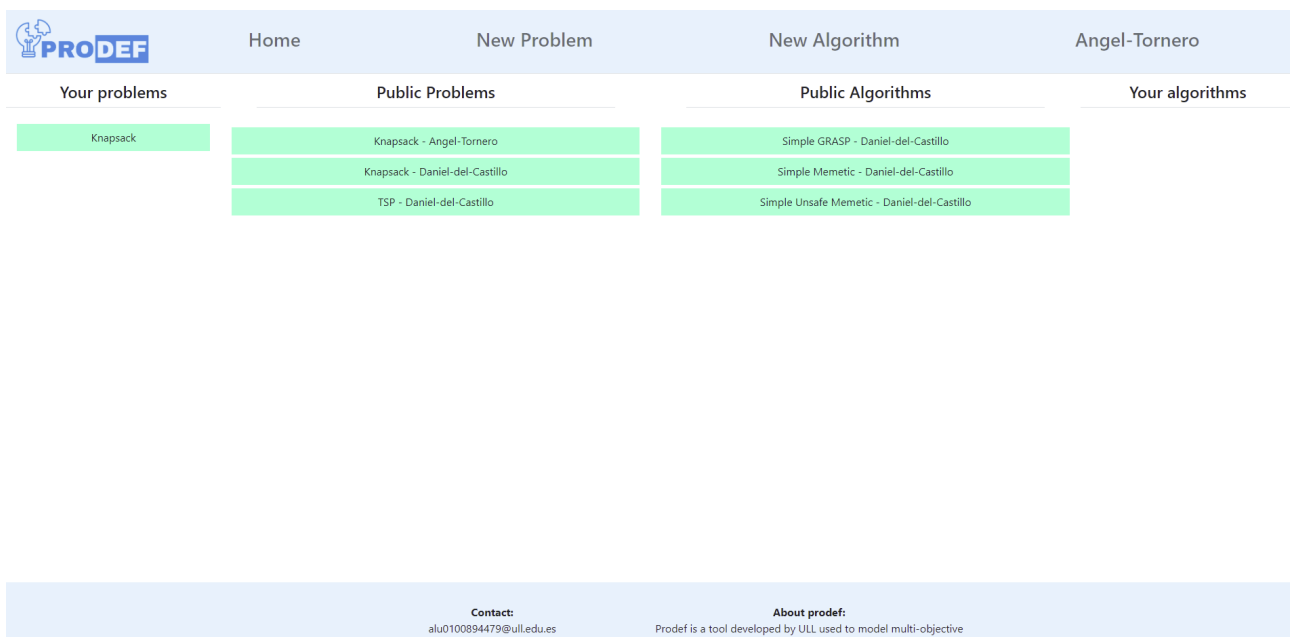


Figura 3.9: Prodef GUI - Página principal

3.5. Aspectos a tener en cuenta

Tras haber realizado la tarea de familiarización con Prodef y haber examinado sus repositorios y el funcionamiento de la herramienta en sí, se ha realizado una lista de particularidades que sería de utilidad tener en cuenta a la hora de comenzar con tareas más avanzadas como las de contenerización y despliegue:

- Actualmente, la organización ULL-prodef y sus repositorios son privados.
- Todos los módulos de Prodef son paquetes *npm*, por lo que es un sistema clave para esta herramienta.
- Se quiere desplegar la última versión de Prodef para dar soporte a la definición de algoritmos. Esto significa que se utilizará el resolutor de Rust y, por tanto, requerirá un mínimo de documentación sobre el gestor de paquetes de Rust [14].
- Prodef funciona gracias a tres servidores que deben estar activos:
 - *Front-end*
 - *Back-end* (que se comunica con una base de datos no relacional)
 - Resolutor
- Cuando se quiere resolver una instancia de un problema desde la GUI, el front-end envía la petición al back-end y este a su vez manda la ejecución al resolutor. La dirección y puerto tanto del back-end como del resolutor están definidas de manera estática en el código utilizando la IP 127.0.0.1. Esto hace funcionar Prodef de manera local, pero dará problemas a la hora de contenerizar la herramienta, ya que esas direcciones se deberán sustituir, o bien por la IP del contenedor, o bien por la IP del servidor en el que se despliegue Prodef.

Capítulo 4

Contenerización de Prodef

Es necesario contenerizar tres componentes de Prodef: resolutor, back-end y front-end. Se sabe que el front-end necesita la dirección IP del back-end para conectarse con la base de datos, y a su vez el back-end necesita la dirección IP del resolutor para realizar las peticiones correspondientes.

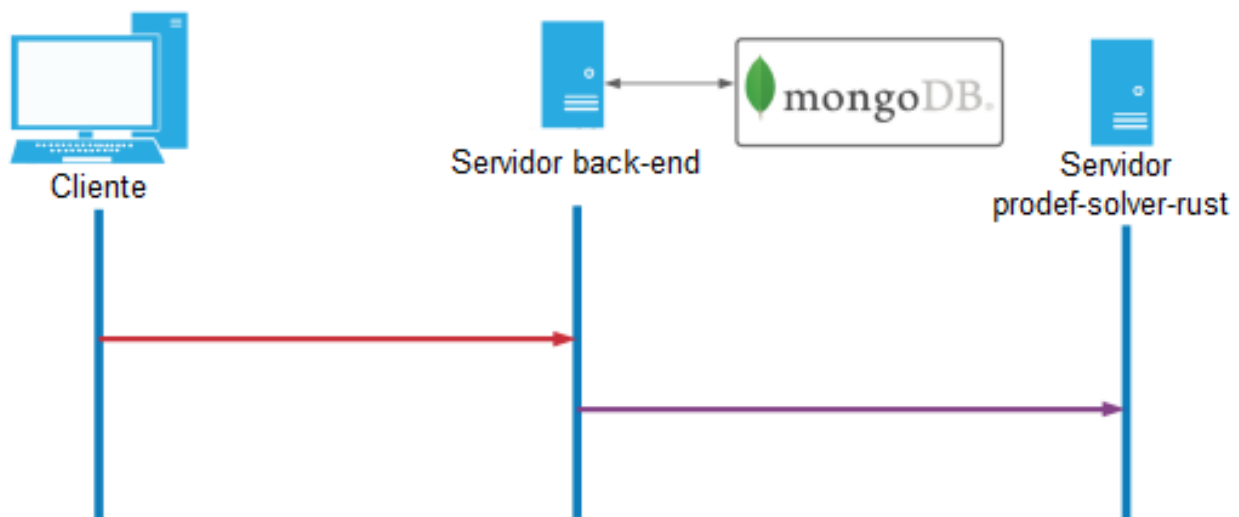


Figura 4.1: Flujo de peticiones

En el Capítulo 3 se mencionó que tanto el front-end como el back-end almacenan estáticamente en el código las direcciones IP, por lo que más adelante habrá que modificar el código con el fin de parametrizarlas. Teniendo en cuenta el flujo de peticiones de Prodef, el orden en el que se deben crear los contenedores es el siguiente:

1. Resolutor (Rust)
2. Back-end, al cual se le pasa la dirección IP del servidor del resolutor
3. Front-end, que recibe la dirección IP del back-end

4.1. Resolutor (Rust)

Respetando el flujo de peticiones que lleva a cabo Prodef (Figura 4.1), se ha decidido crear primero la imagen Docker del resolutor de Rust ya que dicho servidor no depende de que ningún otro esté activo. La primera idea ha sido importar la imagen oficial de Ubuntu y recrear el proceso que se llevó a cabo para instalar el resolutor de manera local. Tras una documentación en profundidad sobre el tema [17] [25], se llegó a la conclusión de que esto no era una buena idea por varias razones. Estas son las buenas prácticas que se han tenido en cuenta para utilizar *Docker*:

- Como se mencionó en el Capítulo 2, lo usual es utilizar una imagen base. Por ejemplo, se podría usar la imagen oficial de *Rust* en este caso.
- Se considera una mala practica depender de una imagen sin especificar un *tag* con una versión concreta. Por ejemplo, si se utiliza «FROM rust» implícitamente se está usando la última versión disponible en ese momento, lo que significa que la versión de rust de la imagen va a depender de cuándo se ejecutó la creación de la imagen. Nada garantiza que el resolutor y sus dependencias sigan funcionando en futuras versiones de Rust, por lo que lo ideal sería especificar alguna versión que se sepa que funciona, por ejemplo «FROM rust:1.61.0».
- Uno de los objetivos al crear imágenes es que el tamaño de los contenedores sea el menor posible; por ello, es recomendable utilizar las versiones *slim* de las imágenes base cuando sea posible, en este caso «FROM rust:1.61.0-slim». La versión *slim* contiene las dependencias mínimas necesarias para ejecutar *Rust* y no incluye ningún paquete común.
- Normalmente, cuando se conteneriza una aplicación, se crea el fichero *dockerfile* en la raíz del propio repositorio de la aplicación, en este caso sería dentro de [prodef-solver-rust](#). Esto elimina la necesidad de tener que clonar un repositorio de Git dentro del *Dockerfile*, ya que se podría simplemente hacer un «COPY».
- Los paquetes de *npm* de Prodef son privados y se necesita un fichero *.npmrc* con credenciales para poder descargarlos durante el *npm install*. Ese archivo *.npmrc* debe ser tratado como un secreto, nunca debería guardarse en la imagen de Docker final porque cualquier usuario podría ver esas credenciales.
- Hay una estrategia muy común a la hora de crear imagenes de Docker que evita que este tipo de secretos acaben en la imagen final. La estrategia se llama *multi-stage builds*. La idea es crear dos imágenes (stages), una que solo se usa para el build (donde se ejecuta *npm install* y por tanto se necesita el archivo *.npmrc*) y otra que solo contenga el artefacto final (los archivos JS ya transpilados). En Docker es posible copiar archivos de un stage a otro. Haciendo esto, nos aseguramos de que la imagen final no va a contener el archivo *.npmrc* ni cualquier otro archivo que no sea estrictamente necesario para ejecutar el resolutor, optimizando así la imagen.
- Construir la imagen entera como *root* puede suponer un problema de seguridad ya que existen técnicas para escapar de un contenedor manteniendo privilegios *root*. Para evitar esta vulnerabilidad, lo mejor es crear un nuevo usuario (que no pertenezca al grupo “*sudo*”) y que el contenedor se ejecute siendo este usuario.

Este es el Dockerfile que construye la imagen deseada:

```
1 # --- STAGE 1 (build) ---
2 FROM node:12.14.1-slim as build
3
4 COPY . ./prodef-solver-rust
5 WORKDIR /prodef-solver-rust
6
7 # prodef solver rust installation
8 RUN npm ci; npm run build
9
10 # --- STAGE 2 (run) ---
11 FROM rust:1.61.0-slim
12 RUN rustup component add rustfmt
13
14 # installing node
15 RUN apt-get update && apt-get install -y curl
16 RUN curl -sL https://deb.nodesource.com/setup_12.x | bash && apt -y install
    nodejs
17
18 RUN useradd -m container
19
20 # copying needful directories from build stage
21 COPY --from=build /prodef-solver-rust/dist /home/container/dist
22 COPY --from=build /prodef-solver-rust/src /home/container/src
23 COPY --from=build /prodef-solver-rust/node_modules/ /home/container/
    node_modules
24 COPY --from=build /prodef-solver-rust/rust /home/container/rust
25
26 RUN chown -R container:container /home/container/rust
27
28 USER container
29 WORKDIR /home/container/rust
30 RUN cargo build --release
31
32 WORKDIR /home/container
33
34 # server start (only when container is initiated)
35 ENTRYPOINT node dist/index.cjs.js
```

1. En la primera *stage* se ha utilizado una imagen *node* para instalar las dependencias con *npm* y transpilar los ficheros.
2. En la segunda *stage* se ha optado por incluir la imagen base de *rust*, en la que se ha instalado manualmente *node* y se ha copiado de la otra escena los directorios y ficheros estrictamente necesarios para ejecutar el servidor. Dentro del directorio *rust/* se ha ejecutado el comando `cargo build --release` para compilar los ficheros de *rust*, siendo la opción `--release` para menor tiempo de cómputo en el proceso de construcción [14]. Como *ENTRYPOINT*, se ha establecido el comando equivalente a `npm run start` definido en el `package.json`.

4.2. Back-end

En cuanto al back-end, ha sido necesario modificar el código para que, tanto la dirección IP del resolutor como las credenciales del repositorio [prodef-secrets](#), obtengan sus valores a través de variables de entorno que se pasan al contenedor a la hora de ejecutarlo. También ha sido necesario comunicarle al back-end (por medio de variables de entorno) cuál es la URL del front-end debido a que, tras realizar la autenticación de GitHub, el servidor lleva a cabo una redirección a dicha URL. Afortunadamente, siempre conoceremos de antemano dicha dirección (por ahora es `http://127.0.0.1:3000`).

Lo primero que se ha hecho es convertir el fichero `prodef-backend.json`, el archivo de [prodef-secrets](#), en un fichero `prodef-backend.env`, el cual se pasará como argumento al comando `docker run` utilizando la opción `--env-file`. A continuación, se han realizado las siguientes modificaciones en el fichero `index.ts` del servidor:

1. Eliminar la lectura del fichero `prodef-backend.json` debido a que este ya no existe. Las credenciales se deben pasar como variables de entorno a los contenedores. Esto implica que en el código se debe hacer uso del objeto `process.env`.
2. Borrar la llamada a la función `childProcess` que ponía en marcha el resolutor, ya que dicho servidor se ejecuta ahora en otro contenedor.
3. Sustituir la dirección del resolutor (que actualmente está definida estáticamente en el código) y en su lugar llamar a las variables de entorno que contienen la IP y el puerto del contenedor de *Rust*. De esta manera, el contenedor del *back-end* y el del resolutor estarán conectados a través de la red de Docker [19].
4. Establecer como URL de redirección de la autenticación por GitHub [7] la dirección que se pase como variable de entorno al contenedor.

Tras haber realizado estas modificaciones en el código, se ha procedido a realizar el último paso que es la generación de la imagen del servidor del *back-end* (manteniendo las buenas prácticas explicadas en el punto anterior).

```
1 FROM node:16.14.2-slim
2
3 RUN useradd -ms /bin/bash container
4 COPY . /home/container/prodef-gui-backend
5 WORKDIR /home/container/prodef-gui-backend
6
7 RUN chown -R container:container /home/container
8 USER container
9
10 # prodef gui backend installation
11 RUN npm install typescript
12 RUN npm ci
13
14 ENTRYPOINT npm run start
```

4.3. Front-end

La contenerización del front-end ha sido el paso más complejo ya que adicionalmente ha requerido documentación acerca de *React* y *Nginx*.

Tras un estudio más en profundidad de la implementación actual del front-end de Prodef y de *React*, se llegó a la conclusión de que estaba pensado únicamente para utilizarlo como un servidor de desarrollo [27]. Esta información se deduce por el hecho de que el `package.json` contenía el atributo "proxy" cuyo valor era la URL del back-end (`http://localhost:5000`). Esto es de utilidad cuando se hacen peticiones *GET* en las que no se especifica la URL, cogiendo por defecto la especificada en el `package.json`. Sin embargo, dicha técnica solo funciona cuando se construye la aplicación de *React* con `npm run start`, es decir, en modo «desarrollo». Esta solución no ha sido de interés, puesto que se tendría que haber establecido como *ENTRYPOINT* de la imagen el comando `npm run start` lo cual, además de requerir recursos como tiempo y almacenamiento, resulta innecesario teniendo en cuenta que el objetivo es desplegar simplemente una serie de ficheros *HTML*, *CSS* y *Javascript* estáticos. En su lugar, se ha optado por generar la aplicación usando `npm run build` para obtener la versión de «producción».

La alternativa a utilizar la propiedad "proxy" es especificar la URL en todos los puntos del código en los que se realice alguna petición al *back-end* (concretamente, en el código que genera la aplicación de *React*). Esto se ha llevado a cabo utilizando nuevamente variables de entorno, con la diferencia de que esta vez se pasan en tiempo de construcción de la imagen y no al crear el contenedor. Esto no es un problema ya que siempre conocemos de antemano la URL del back-end (en este caso `http://127.0.0.1:5000`). Se ha procedido entonces a realizar los siguientes cambios en el código:

- Eliminar la propiedad "proxy" del `package.json`.
- Buscar todos los puntos del código en los que se hace una petición *GET* y añadir la URL del back-end a través de variables de entorno. Cabe mencionar que para que *React* reconozca variables de entorno en el código fuente, estas deben comenzar por `REACT_APP_` [28].
- Parametrizar el enlace del back-end que lleva a la autenticación por GitHub ya que también está almacenado estáticamente en el código.

Dicho esto, la estrategia de contenerización ha sido utilizar una *multi-stage build* [4]:

- En la primera etapa se utiliza una imagen de *node* en la que se construye la aplicación *React* utilizando todos los ficheros necesarios (incluyendo el `.npmrc` para instalar los paquetes privados).
- En la segunda etapa utilizaremos el servidor web *Nginx* a través de su imagen oficial, al igual que se empleó en el Capítulo 2. No ha sido necesario utilizar una imagen de *node* porque gracias a la primera *stage*, ya se tienen todos los ficheros estáticos transpilados, que es lo único necesario realmente.

El servidor *Nginx* del contenedor [24] se ha configurado según este fichero:

```
1 server {
2     listen 3000 default_server;
3     root /var/www/html;
4     index index.html;
5     server_name 127.0.0.1;
6     location / {}
7 }
```

Este es el `dockerfile` que construye la imagen del front-end, el cual requiere que se le pase como argumento (en tiempo de construcción de la imagen) la IP y el puerto del back-end:

```
1 # --- STAGE 1 (build) ---
2 FROM node:16.14.2-slim as build
3
4 ARG BACKEND_IP
5 ARG BACKEND_PORT
6
7 COPY . ./prodef-gui-frontend
8 WORKDIR /prodef-gui-frontend
9
10 # prodef gui frontend installation and build
11 RUN npm ci
12 RUN REACT_APP_BACKEND_IP=$BACKEND_IP REACT_APP_BACKEND_PORT=$BACKEND_PORT npm
    run build
13
14 # --- STAGE 2 (run) ---
15 FROM nginx:1.19.0-alpine
16 COPY --from=build /prodef-gui-frontend/build/. /var/www/html
17 COPY --from=build /prodef-gui-frontend/default.conf /etc/nginx/conf.d/default.
    conf
18 ENTRYPOINT ["nginx", "-g", "daemon off;"]
```

4.4. Prodef docker-compose

Una vez construidas las tres imágenes, este es el docker-compose que reúne todas las características mencionadas en las secciones anteriores:

```
1 version: "3.0"
2 services:
3   solver:
4     build:
5       context: ../prodef-solver-rust
6       container_name: solver_container
7       image: prodef-solver-rust
8   backend:
9     build:
10      context: ../prodef-gui-backend
11      container_name: backend_container
12      image: prodef-gui-backend
13      ports:
14        - "5000:5000"
15      env_file:
16        - prodef-backend.env
17      environment:
18        - SOLVER_IP=solver_container
19        - SOLVER_PORT=8046
20        - URL=http://127.0.0.1
21        - FRONTEND_PORT=3000
22   frontend:
23     build:
24       context: ../prodef-gui-frontend
25       args:
26         - URL=http://127.0.0.1
27         - BACKEND_PORT=5000
28       image: prodef-gui-frontend
29     ports:
30       - "3000:3000"
```

Se ha creado un nuevo repositorio en la organización de ULL-prodef, llamado [prodef-docker-compose](#), que incluye el fichero `docker-compose.yml` que se acaba de mostrar.

Al ejecutar el comando `docker-compose up --build`, se construyen las imágenes utilizando la información de construcción de los servicios, se crean los tres contenedores y la herramienta será utilizable completamente accediendo desde el navegador a la URL `http://127.0.0.1:3000`. La Figura 4.2 es un diagrama resumen del docker-compose, explicando de qué manera fluye la información en las diferentes etapas (*build* y *run*) y cómo se interconectan los contenedores.

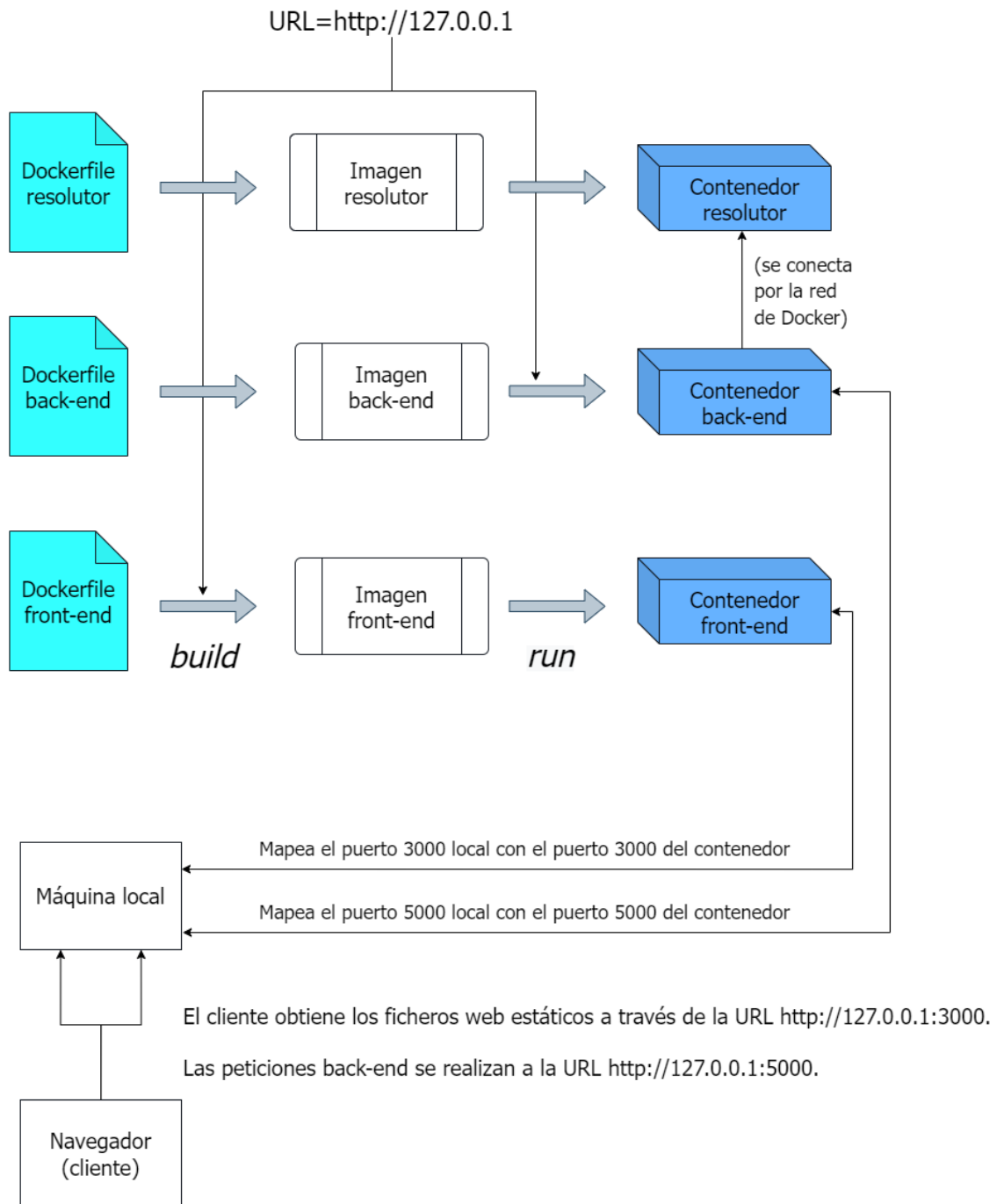


Figura 4.2: Diagrama del funcionamiento del docker-compose de Prodef

Capítulo 5

Despliegue de Prodef

Para este proceso se ha utilizado el *droplet* de *DigitalOcean* obtenido (véase el Capítulo 2). La idea es configurar un despliegue de manera manual, que una vez realizado, se recrearía mediante código *Terraform*. El manejo de la máquina virtual se ha llevado a cabo a través de *SSH* gracias al entorno de desarrollo integrado *Visual Studio Code*.

5.1. Despliegue manual

El despliegue manual ha consistido en recrear la configuración que se llevó a cabo localmente para poner en marcha el *docker-compose* de Prodef. Se ha querido dejar constancia de todos los pasos seguidos en el proceso de despliegue manual de la herramienta. Cabe destacar que se ha partido de un servidor *Ubuntu* recién creado.

1. Generar una clave pública mediante el comando `ssh-keygen` para añadirla a *GitHub*.
2. Establecer un directorio de trabajo para clonar los repositorios `prodef-solver-rust`, `prodef-gui-backend`, `prodef-gui-frontend` y `prodef-docker-compose`.
3. Cambiar la rama de los repositorios `prodef-gui-backend` y `prodef-gui-frontend` a *deployment*, ya que es la que contempla los cambios realizados en el Capítulo 4.
4. Crear un fichero `.npmrc` con un *token* válido y copiarlo en `prodef-solver-rust` y `prodef-gui-frontend` ya que es necesario para instalar las dependencias.
5. Instalar `docker` y `docker-compose` mediante el comando `apt install`.
6. Realizar los siguientes cambios en el `docker-compose.yml`:
 - Cambiar el valor de la variable `URL`, que aparece dos veces y cuyo valor es `http://127.0.0.1`, y sustituir la IP de `localhost` por la IP pública de la máquina virtual de despliegue, siendo el nuevo valor `http://178.128.198.188`.
 - Redirigir el puerto 3000 del contenedor del front-end al puerto 80 de la máquina virtual. Al ser este el puerto por defecto de *http* [8], se podrá acceder a la página de Prodef simplemente escribiendo la IP (sin el puerto) en el navegador.
7. El cambio de IP conllevó un desajuste en la autenticación por *GitHub*, ya que la aplicación *OAuth* [7] tenía establecida la dirección `http://127.0.0.1` como URL de autorización y ahora debe ser `http://178.128.198.188`, que es la que recibirá

el código del back-end por variable de entorno. No se dispone de acceso a dicha aplicación OAuth para modificar ese campo, por lo que se ha optado por crear una nueva a través de la página de GitHub, en el apartado *Developer Settings*.

8. Crear el fichero `prodef-backend.env` en el directorio `prodef-docker-compose/` con la información del repositorio `prodef-secrets` (claves del *back-end*). Como en el paso anterior se creó una nueva aplicación OAuth, se tuvo que actualizar los valores de *clientID* y *clientSecret*.
9. Cambiar el fichero de configuración de *Nginx* para sustituir el valor de `server_name`; antes era `127.0.0.1` y ahora es `178.128.198.188`.
10. Ejecutar el comando `docker-compose up -build`, lo cual crea la imagen de cada servicio del `docker-compose` que tenga la propiedad *build* (todos en este caso) y posteriormente pone en marcha los tres contenedores. Una vez realizado este último paso, la herramienta comenzó a ser accesible a través de la URL `http://178.128.198.188`.

5.2. Nombre de dominio

Se ha decidido adquirir un nombre de dominio por dos razones principales:

- Un nombre de dominio mejora la experiencia de usuario.
- Utilizar un nombre de dominio nos libraría de la tarea de modificar el `docker-compose` cada vez que se cambie el servidor de despliegue.

Para obtener el nombre, se ha optado por la web [Namecheap](#). *Namecheap* es un registrador de nombres de dominio acreditado por la ICANN (*Internet Corporation for Assigned Names and Numbers*) [15]. Lo primero que se ha hecho tras registrarse en la página ha sido comprobar la validez del nombre *prodef* en la web (Figura 5.1).

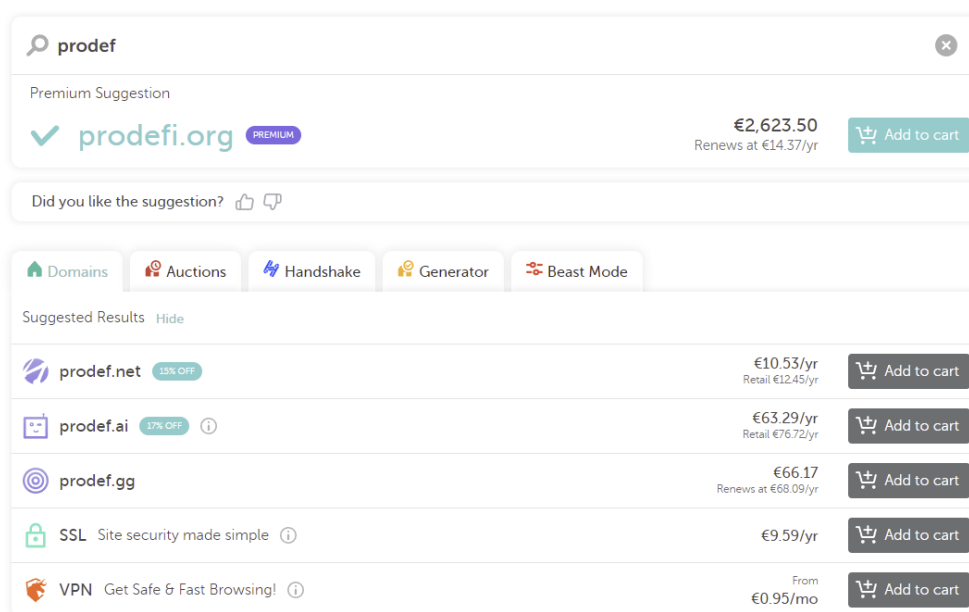


Figura 5.1: Validez del nombre de dominio *prodef*

Tras valorar todas las posibilidades, se ha elegido como nombre de dominio *prodef.pro* por ser fácil de recordar además de resultar económico (3,97 €/año). Una vez realizada la compra y autenticada la cuenta, se pudo comenzar a configurar el nombre. *DigitalOcean* ofrece la posibilidad de gestionar desde la propia página los nombres de dominio que se poseen, lo cual se ha configurado realizando los siguientes pasos:

1. Acceder al menú *Domains/DNS* en *DigitalOcean* y añadir el dominio que hemos adquirido (Figura 5.2).
2. En el panel de gestión de dominios en Namecheap, establecer como DNS (sistema de nombres de dominio) un DNS personalizado (*Custom DNS*) y añadir tres servidores DNS de *DigitalOcean* cuyos nombres son *ns1.digitalocean.com*, *ns2.digitalocean.com* y *ns3.digitalocean.com* (Figura 5.3).

Networking

Domains Reserved IPs Load Balancers VPC Firewalls PTR records

Add a domain

Enter a domain that you own below and start managing your DNS within your DigitalOcean account.

Enter domain * first-project Add Domain

Domains

Domain	Directs to
prodef.pro 1A / 3 NS	prodef FRA1 / 178.128.198.188

Figura 5.2: DNS de DigitalOcean

prodef.pro

Domain Products Sharing & Transfer Advanced DNS

STATUS & VALIDITY ? ACTIVE Jun 29, 2022 - Jun 29, 2023 AUTO-RENEW ADD YEARS

WithheldforPrivacy ? PROTECTION Jun 29, 2022 - Jun 29, 2023 AUTO-RENEW SHOW DETAILS

PremiumDNS ? Enable PremiumDNS protection in order to switch your domain to our PremiumDNS platform. With our PremiumDNS platform, you get 100% DNS uptime and DDoS protection at the DNS level. BUY NOW

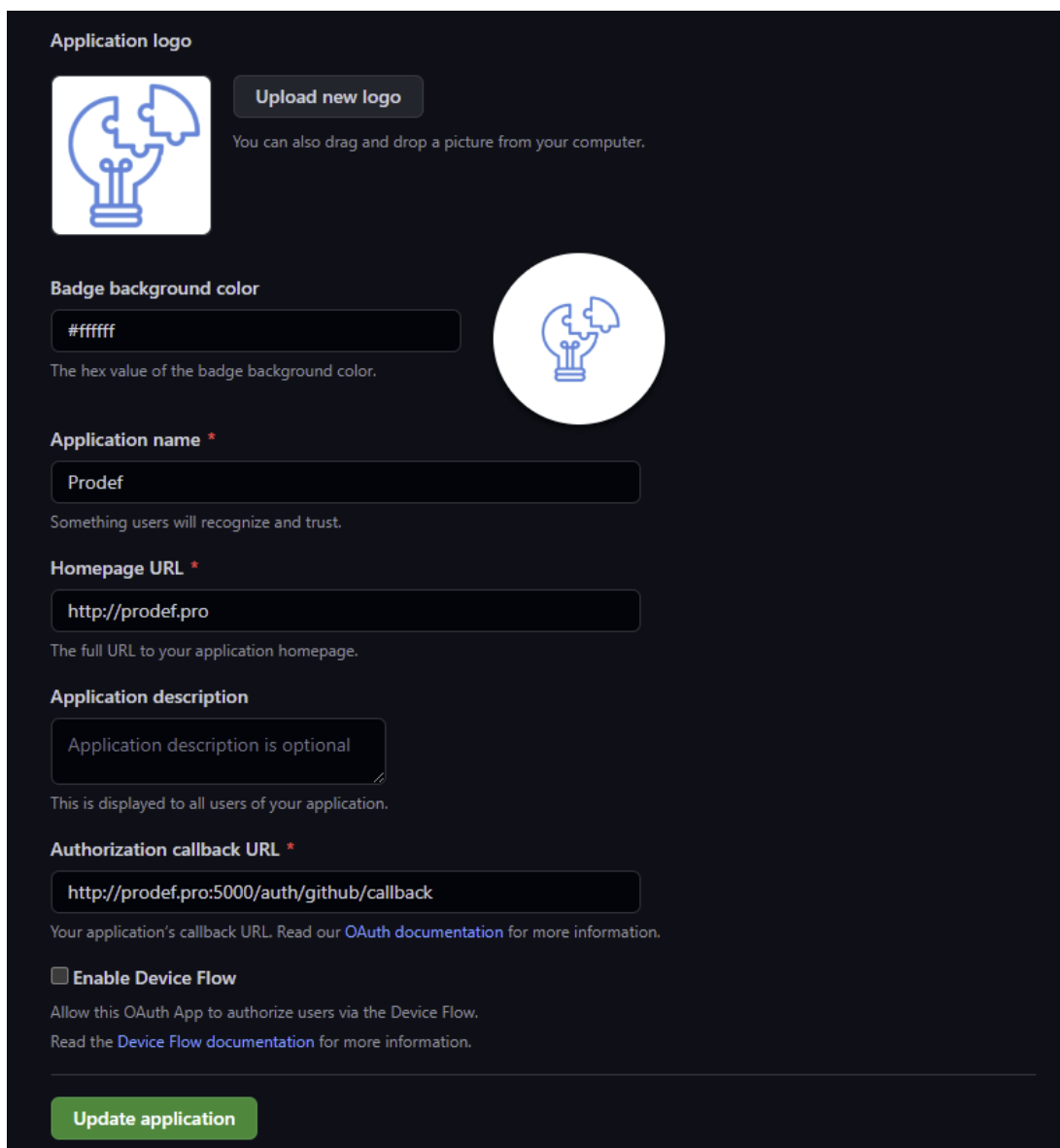
NAMESERVERS ? Custom DNS

ns1.digitalocean.com
ns2.digitalocean.com
ns3.digitalocean.com
ADD NAMESERVER

Figura 5.3: Configuración de servidor DNS en Namecheap

Con esto, comenzó a ser posible gestionar el nombre *prodef.pro* desde *DigitalOcean*, lo cual aumenta la comodidad al tener todo centralizado en un mismo entorno. El último paso para configurar el nombre del dominio ha sido agregar un registro de tipo A, que se utiliza para direcciones de tipo IPv4 e indica a donde se debe dirigir el dominio. En este caso se indicó como dirección *178.128.198.188*, y gracias al cambio de redirección de puerto que se realizó en el *docker-compose* durante el despliegue manual (Sección 5.1), esa URL es suficiente para entrar en la página ya que por defecto el navegador accede al puerto 80 del servidor.

Al contrario de lo que había parecido en un primer momento, esto no era todo; tanto el back-end como la aplicación OAuth entendían que la página principal era *http://178.128.198.188* y no *http://prodef.pro*, y como el *back-end* solo envía información de la base de datos si se accede por la URL especificada, se ha tenido que actualizar nuevamente tanto la URL en la aplicación OAuth de GitHub (Figura 5.4) como las variables URL en el *docker-compose*.



The image shows a dark-themed configuration page for a GitHub OAuth application. The page includes several sections with input fields and labels:

- Application logo:** A lightbulb icon with puzzle pieces inside. A button labeled "Upload new logo" is next to it. Below the button, it says "You can also drag and drop a picture from your computer."
- Badge background color:** A text input field containing "#ffffff". Below it, it says "The hex value of the badge background color." To the right is a circular badge with the lightbulb icon on a white background.
- Application name *:** A text input field containing "Prodef". Below it, it says "Something users will recognize and trust."
- Homepage URL *:** A text input field containing "http://prodef.pro". Below it, it says "The full URL to your application homepage."
- Application description:** A text area containing "Application description is optional". Below it, it says "This is displayed to all users of your application."
- Authorization callback URL *:** A text input field containing "http://prodef.pro:5000/auth/github/callback". Below it, it says "Your application's callback URL. Read our [OAuth documentation](#) for more information."
- Enable Device Flow:** A checkbox that is currently unchecked. Below it, it says "Allow this OAuth App to authorize users via the Device Flow. Read the [Device Flow documentation](#) for more information."

At the bottom of the page is a green button labeled "Update application".

Figura 5.4: Información de la aplicación OAuth

Este es el `docker-compose.yml` final:

```
1 version: "3.0"
2 services:
3   solver:
4     build:
5       context: ../prodef-solver-rust
6     container_name: solver_container
7     image: prodef-solver-rust
8   backend:
9     build:
10      context: ../prodef-gui-backend
11     container_name: backend_container
12     image: prodef-gui-backend
13     ports:
14       - "5000:5000"
15     env_file:
16       - prodef-backend.env
17     environment:
18       - SOLVER_IP=solver_container
19       - SOLVER_PORT=8046
20       - URL=http://prodef.pro
21       - FRONTEND_PORT=80
22   frontend:
23     build:
24       context: ../prodef-gui-frontend
25     args:
26       - URL=http://prodef.pro
27       - BACKEND_PORT=5000
28     image: prodef-gui-frontend
29     ports:
30       - "80:3000"
```

Para tener constancia de estos cambios, ha sido necesario volver a ejecutar el comando `docker-compose up --build` para reconstruir las imágenes del back-end y el front-end con el valor de la URL actualizado y ejecutar nuevos contenedores. Una vez se han vuelto a poner en marcha los tres servicios, dejó de ser necesario referirse a la página por la IP del servidor y comenzó a estar activo el enlace <http://prodef.pro>.

5.3. Despliegue con Terraform

En el Capítulo 2 se mencionó que Terraform es capaz de conectarse a la API de un proveedor de servicio de nube para desplegar una infraestructura mediante código. En primer lugar, se ha creado un fichero llamado `main.tf` cuyo objetivo es indicar la API a la que se quiere conectar. Cuando se ejecute el comando `terraform init`, Terraform descargará los ficheros necesarios para conectarse a la API indicada. En este caso, se ha elegido DigitalOcean porque es la conocida. Este es el fichero en cuestión:

```
1 terraform {
2   required_providers {
3     digitalocean = {
4       source = "digitalocean/digitalocean"
5       version = "~> 2.0"
6     }
7   }
8 }
```

Una vez se ha inicializado el directorio de trabajo de Terraform, primero debe indicarse quién va a proporcionar esos recursos. Para establecer una plataforma como proveedor, es necesario un *token* de acceso personal a la API de esta. En el caso de DigitalOcean, se puede crear a través de su página web [16]. Para introducir el *token* de manera segura, se ha utilizado una variable de entorno. Terraform solamente reconoce las variables de entorno que empiezan por `TF_VAR_`, por lo que se ha exportado una variable de entorno llamada `TF_VAR_digitalocean_token` y se ha establecido como valor el *token* que se ha creado. Se ha creado un nuevo fichero llamado `provider.tf` siguiendo esta información:

```
1 variable "digitalocean_token" {}
2
3 # Configure the DigitalOcean Provider
4 provider "digitalocean" {
5   token = "${var.digitalocean_token}"
6 }
```

Ya se ha establecido el proveedor de la infraestructura; ahora se puede empezar a definir los recursos requeridos. Los recursos necesarios para esta tarea son los mismos que se han configurado a mano en la Sección 5.1 y 5.2: una clave pública, un servidor virtual (o *droplet*) y un nombre de dominio. Se ha creado un fichero `.tf` para definir cada uno de estos recursos.

5.3.1. Clave pública

Para permitir la comunicación entre el servidor virtual que se va a crear y la máquina local, se ha tenido que compartir la clave pública de la máquina local. De esta forma, más adelante se podrán definir algunos recursos locales que se desean enviar al *droplet*. La función `file` en Terraform recoge como argumento la ruta de un fichero y devuelve su contenido. Este es el contenido del fichero `01_ssh_key.tf`:

```
1 # Export the SSH key
2
3 resource "digitalocean_ssh_key" "public_key_prodef" {
4   name = "public_key_prodef"
5   public_key = "${file("id_rsa.pub")}"
6 }
```

5.3.2. Droplet

Definir el *droplet* ha sido un poco más complejo. Lo primero ha sido definir las características y recursos físicos de la máquina tal y como se decidió en la Sección 2.2. La única diferencia en las decisiones es que esta vez se optó por una máquina con el doble de CPU y el doble de RAM, debido a que el proceso de construcción de la aplicación *React* se detenía por exceder el tiempo máximo de ejecución.

A continuación, se declararon dos proveedores locales de tipo “fichero”. Esto significa que hay dos recursos que la máquina local le va a enviar a la máquina virtual en el proceso de creación. Los ficheros que se desea enviar son:

- `.npmrc`: necesario para instalar las dependencias de Prodef.
- `prodef-backend.env`: requerido para el funcionamiento del *back-end*.

Para ello, no solo se debe indicar la ruta de origen y de destino, sino la forma de conexión que va a tener la máquina local con el servidor. En este caso, se realizará por *ssh* ya que la máquina contará con nuestra clave pública.

Por último, se declaró un proveedor de tipo “ejecución remota” (el cual necesita también especificación sobre la forma de conexión). Este tipo de recurso cuenta con un atributo `inline`, el cual se debe definir como una lista de *strings*, cada una de estas conteniendo un comando. Esta cadena de comandos es la que se ejecutará en el *droplet* al ser creado. Escribir la Sección 5.1 fue de mucha ayuda en este proceso ya que es dicho proceso el que se debe recrear en este paso.

En el Apéndice A se puede ver el fichero que contiene toda esta información, llamado `02_droplet.tf`.

5.3.3. Nombre de dominio

Realmente se han creado dos recursos para configurar el nombre de dominio, pero se han agrupado en un solo apartado. En la Sección 5.2 se explicó la forma que tiene DigitalOcean de gestionar los nombres de dominio; es ese el proceso que se va a recrear en Terraform. Crear un recurso de tipo dominio es una tarea muy trivial, pues solo requiere rellenar el atributo `name` con el propio nombre de dominio. En este caso, se le ha dado el valor de “`prodef.pro`” ya que es el nombre que se ha registrado para este trabajo. Una vez definido el dominio, se va a definir el segundo recurso, que es un registro del nombre de dominio. El registro será del tipo A y su dirección será la IPv4 del servidor virtual que se va a crear. Esta dependencia debe ser indicada también. Este es el último fichero de configuración de la infraestructura, llamado `03_dns.tf`:


```

1  # Domain name
2
3  resource "digitalocean_domain" "prodef" {
4    name = "prodef.pro"
5  }
6
7  resource "digitalocean_record" "www" {
8    domain = "${digitalocean_domain.prodef.name}"
9    type = "A"
10   name = "@"
11   ttl = "3600"
12   value = "${digitalocean_droplet.prodef_server.ipv4_address}"
13   depends_on = [
14     digitalocean_droplet.prodef_server
15   ]
16 }
17

```

Para crear la infraestructura, se deben introducir los siguientes comandos tal y como se describió en la Sección 2.3 :

```

terraform plan
terraform apply

```

Actualmente Prodef está desplegado en una infraestructura creada a través de Terraform gracias al plan definido en este capítulo. La Figura 5.5 muestra un pequeño resumen a modo de esquema del proceso explicado: desde que se escriben los ficheros de configuración hasta que la infraestructura está montada

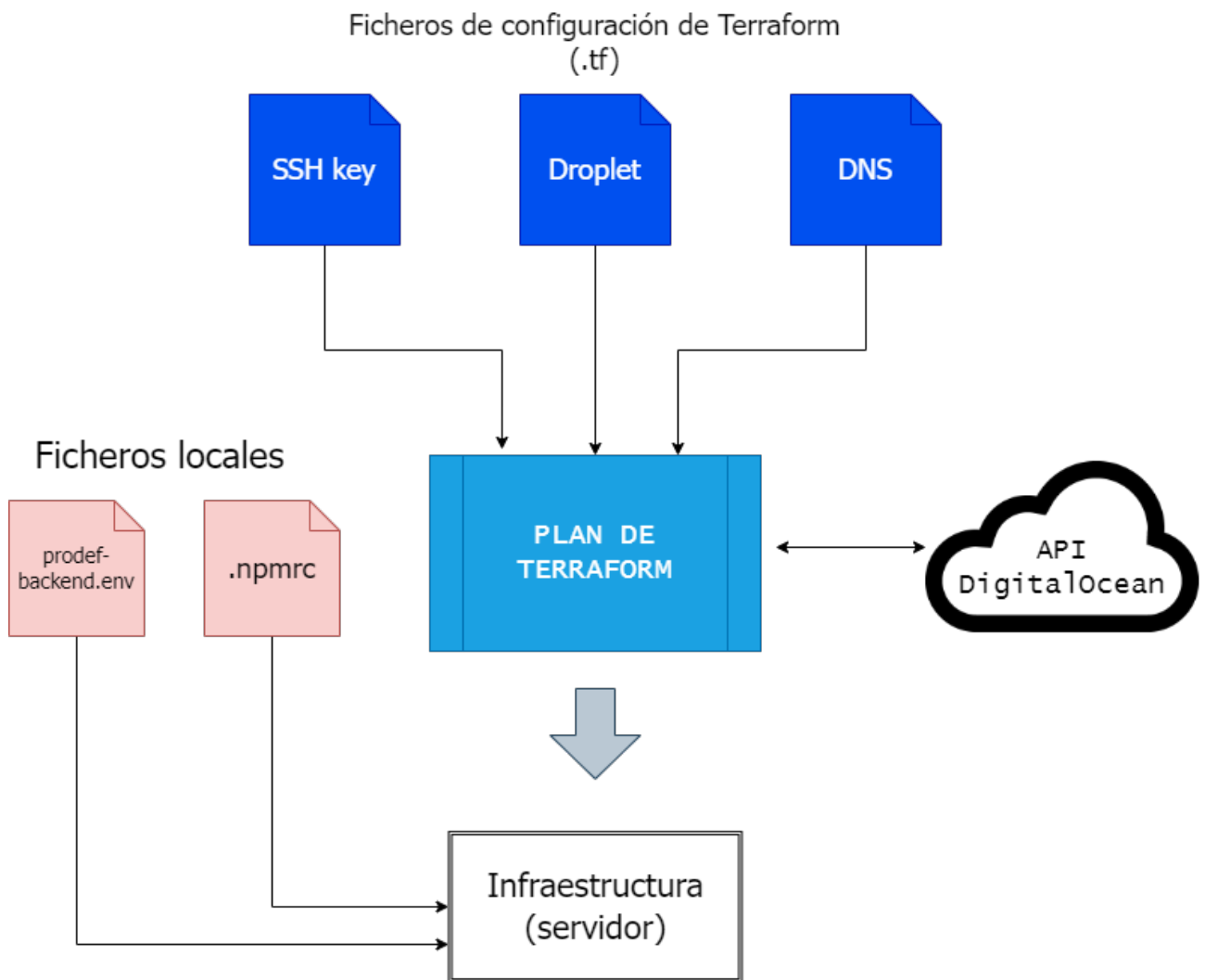


Figura 5.5: Esquema de la creación de la infraestructura de Prodef

Capítulo 6

Conclusiones y líneas futuras

Durante este trabajo se han investigado diferentes alternativas para que la herramienta Prodef pueda llegar a su público objetivo. Prodef es una herramienta para la resolución de problemas de optimización combinatoria. Esta herramienta permite a un usuario modelar sus propios problemas y definir algoritmos a través de una interfaz gráfica basada en bloques, y posteriormente obtener una solución. Después del trabajo realizado en este Trabajo de Fin de Grado, Prodef cuenta con una infraestructura para ofrecerla como un servicio en la nube. Para diseñar esta arquitectura de despliegue, se ha llevado a cabo una documentación en profundidad acerca de las tecnologías y plataformas más empleadas para este tipo de trabajo. Primero, se ha contenerizado cada microservicio de la herramienta utilizando Docker. Asimismo, se ha realizado una configuración en Docker-compose capaz de poner en marcha todos los contenedores de Prodef simultáneamente. Para esta tarea ha sido necesario realizar cambios en el código tanto del back-end como del front-end. También ha sido necesario contratar un proveedor de servicios de nube como DigitalOcean. Por último, se ha desarrollado un plan de Terraform capaz de crear un servidor virtual nuevo que, automáticamente, realiza todos los pasos necesarios para poner el marcha el Docker-compose de Prodef, consiguiendo de esta forma el despliegue del servicio.

6.1. Conclusiones

La consistencia de los resultados compensan la alta curva de aprendizaje de las técnicas y conceptos que se utilizan para ofrecer un Software Como Servicio (SaaS). Este modelo proporciona la herramienta como un servicio disponible en Internet, de forma que el usuario no tenga que instalar ningún software ni preocuparse de aspectos técnicos para poder hacer uso del servicio deseado. Con SaaS es posible ofrecer a los usuarios finales aplicaciones relativamente complejas de una forma casi transparente, sin necesidad de adquirir, instalar, actualizar o mantener ningún tipo de hardware, middleware o software. De esta forma, aplicaciones empresariales sofisticadas quedan al alcance de organizaciones que no cuentan con grandes recursos para adquirir infraestructuras ni implementar soluciones a medida.

Como resultado de este trabajo no solo se ha establecido un plan de despliegue portable a cualquier proveedor de servidores virtuales, sino que también se ha adaptado el código de Prodef para futuras tareas de este ámbito. Además, se ha dejado operativo un enlace con nombre de dominio para acceder a Prodef desde cualquier navegador:

<http://prodef.pro>.

A título personal, la realización de este trabajo me ha abierto las puertas a campos totalmente nuevos dentro de la Ingeniería Informática, como son los modelos de distribución de *software* y las tecnologías de despliegue de servicios en la nube. Al haber realizado el itinerario de Computación, por ahora estaba más enfocado en el desarrollo y no había cursado asignaturas que me pudiesen introducir en este ámbito. Por esta razón, creo que este Trabajo de Fin de Grado ha sido un complemento perfecto a todas las demás competencias adquiridas en la carrera. Ha provocado cierta incertidumbre el hecho de realizar un Trabajo de Fin de Grado sobre un área de conocimiento desde la cual se partía prácticamente desde 0, pero considero que una de las mayores cualidades que debe tener un Ingeniero Informático es la capacidad para adaptarse a cualquier tipo de trabajo dentro de la Informática.

6.2. Líneas de trabajo futuras

A continuación, se detalla una lista de posibles líneas de trabajo para continuar mejorando Prodef:

- **Despliegue de Prodef a gran escala utilizando un sistema como *Kubernetes*.** Tras la realización de este trabajo, se tiene una infraestructura para desplegar Prodef automatizadamente gracias a Terraform. La idea no es cargar Prodef con múltiples ejecuciones ahora mismo, por lo que esta opción sirve de momento. En un futuro, se puede optar por la creación de un clúster (*Kubernetes*) donde se desplieguen resolutores en función de la demanda que exista en cada momento.
- **Usabilidad, accesibilidad y experiencia de usuario.** Sería interesante investigar maneras de mejorar la usabilidad, accesibilidad y experiencia de usuario. Una buena opción es recopilar datos a través de un cuestionario para los usuarios que prueben la herramienta (debería ser una gran cantidad de personas). Asimismo, se podría ofrecer en el front-end explicaciones sobre como usar cada bloque tanto en problemas como en algoritmos. También estaría bien mejorar la forma en la que se ofrece la solución a los problemas, haciendo que esta sea más sencilla de entender.
- **Reconocimiento del lenguaje natural.** Esta línea de trabajo es la que quizás podría verse en un futuro menos inmediato. Podría comenzarse por simplemente un estudio de la viabilidad de un sistema inteligente capaz de definir un problema tal y como los representa Prodef (en un formato JSON con ciertos campos con expresiones en ProdefLang) dándole como única entrada un texto en lenguaje natural. Por ejemplo, para la restricción del problema de la mochila, un texto que diga *“La suma de los pesos de cada objeto dentro de la solución no debe exceder el valor del peso máximo”* debería traducirse en la siguiente expresión en ProdefLang:

```
sum x[i]*item[i].weight over i=(1:N) <= MaxWeight
```

Es una línea de trabajo muy compleja pero, a nivel tecnológico, podría ser uno de los mayores logros de Prodef. También llevaría a la herramienta casi totalmente a lograr su objetivo de acercar los algoritmos de optimización bioinspirados a personas sin

experiencia, pues estas podrían simplemente explicar sus problemas en lenguaje natural, sin necesidad de entender la interfaz ni la notación matemática.

Capítulo 7

Summary and Conclusions

During this work, different alternatives have been investigated so that the Prodef tool can reach its target audience. Prodef is a tool for solving combinatorial optimization problems. This tool allows an user to model his own problems and define algorithms through a block-based graphical interface, and subsequently obtain a solution. After the work done in this Final Degree Project, Prodef has an infrastructure to offer it as a cloud service. To design this deployment architecture, an in-depth documentation of the technologies and platforms most commonly used for this type of work has been carried out. First, each microservice of the tool has been containerized using Docker. Also, a configuration has been made in Docker-compose capable of launching all Prodef containers simultaneously. For this task it has been necessary to make changes in the code of both the back-end and front-end. It has also been necessary to contract a cloud service provider such as DigitalOcean. Finally, a Terraform plan has been developed to create a new virtual server that automatically performs all the necessary steps to launch Prodef's Docker-compose, thus achieving the deployment of the service.

7.1. Conclusions

The consistency of the results compensates for the high learning curve of the techniques and concepts used to offer Software as a Service (SaaS). This model provides the tool as a service available on the Internet, so that the user does not have to install any software or worry about technical aspects in order to make use of the desired service. With SaaS it is possible to offer end users relatively complex applications in an almost transparent way, without the need to purchase, install, update or maintain any hardware, middleware or software. In this way, sophisticated business applications are within the reach of organizations that do not have the resources to acquire infrastructure or implement customized solutions.

As a result of this work, not only has a deployment plan been established that is portable to any virtual server provider, but also the Prodef code has been adapted for future tasks in this area. In addition, a domain name link has been made operational to access Prodef from any browser: <http://prodef.pro>.

On a personal note, the completion of this work has opened the doors to completely new fields within Computer Engineering, such as software distribution models and cloud service deployment technologies. Having completed the Computing itinerary, I was more

focused on development and I had not taken subjects that could introduce me to this field. For this reason, I believe that this Final Degree Project has been a perfect complement to all the other skills acquired in the degree. It has caused some uncertainty the fact of doing a Final Degree Project on an area of knowledge from which I started practically from scratch, but I believe that one of the greatest qualities that a Computer Engineer must have is the ability to adapt to any type of work in Computer Science.

7.2. Lines for future work

The following is a list of possible lines of work to continue improving Prodef:

- **Large-scale deployment of Prodef using a system such as Kubernetes.** After the completion of this work, an infrastructure is in place to deploy Prodef automated thanks to Terraform. The idea is not to load Prodef with multiple executions right now, so this option works for now. In the future, we can consider the creation of a cluster (Kubernetes) where resolvers are deployed depending on the demand that exists at any given time.
- **Usability, accessibility and user experience.** It would be interesting to investigate ways to improve usability, accessibility and user experience. A good option is to collect data through a questionnaire for users who test the tool (it should be a large number of people). Also, explanations on how to use each block on problems and algorithms could be provided on the front-end. It would also be good to improve the way in which the solution to the problems is offered, making it easier to understand.
- **Natural language recognition.** This line of work is the one that could perhaps be envisaged in the less immediate future. One could start by simply studying the feasibility of an intelligent system capable of defining a problem as represented by Prodef (in a JSON format with certain fields with expressions in ProdefLang) by giving it as a single input a natural language text. For example, for the Knapsack problem constraint, a text that reads *“The sum of the weights of each object within the solution must not exceed the value of the maximum weight”* should be translated into the following ProdefLang expression:

```
sum x[i]*item[i].weight over i=(1:N) <= MaxWeight
```

It is a very complex line of work but, on a technological level, it could be one of Prodef’s greatest achievements. It would also lead the tool almost entirely to achieve its goal of bringing bio-inspired optimization algorithms closer to inexperienced people, as they could simply explain their problems in natural language, without needing to understand the interface or the mathematical notation.

Capítulo 8

Presupuesto

En este capítulo se presenta el presupuesto del trabajo realizado. El coste de este proyecto proviene del tiempo que se ha empleado en su desarrollo y de una serie de gastos extra por alquiler de servicio. Para estimar el coste por horas, se ha consultado en varias plataformas el sueldo medio de un Ingeniero *DevOps* y se ha elegido la cifra de 19,40€ por hora.

Nombre	Horas	Coste (€)
Investigación del funcionamiento de Prodef	25	485
Aprendizaje de tecnologías	100	1940
Adaptación del código de Prodef para el despliegue	25	485
Contenerización de Prodef	100	1940
Diseño de la infraestructura de despliegue	40	776
Despliegue y puesta en marcha del servicio	10	196
Total	300	5822

Tabla 8.1: Presupuesto por tiempo trabajado

Nombre	Coste (€)
Registro de dominio prodef.pro	4.09
Costes por alquiler de droplet DigitalOcean	6.00/mes

Tabla 8.2: Gastos extra

Apéndice A

Fichero de configuración de Terraform para definir el droplet de DigitalOcean

```
1 # Droplet creation
2
3 resource "digitalocean_droplet" "prodef_server" {
4   image = "ubuntu-20-04-x64"
5   name = "prodef-tf"
6   region = "fra1"
7   size = "s-2vcpu-2gb"
8   ssh_keys = ["${digitalocean_ssh_key.public_key_prodef.fingerprint}"]
9
10  provisioner "file" {
11    source = "./prodef-backend.env"
12    destination = "/root/prodef-backend.env"
13    connection {
14      type = "ssh"
15      user = "root"
16      host = "${self.ipv4_address}"
17    }
18  }
19
20  provisioner "file" {
21    source = "./.npmrc"
22    destination = "/root/.npmrc"
23    connection {
24      type = "ssh"
25      user = "root"
26      host = "${self.ipv4_address}"
27    }
28  }
29
30  provisioner "remote-exec" {
31    inline = [
32
33      # Docker and Docker compose
34
```

```

35     "sudo apt-get update",
36     "sudo apt-get install -y apt-transport-https ca-certificates curl gnupg-
agent software-properties-common",
37     "curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key
add -",
38     "sudo add-apt-repository \"deb [arch=amd64] https://download.docker.com/
linux/ubuntu $(lsb_release -cs) stable\"",
39     "sudo apt-get update",
40     "sudo DEBIAN_FRONTEND=noninteractive apt-get install -y docker-ce docker
-ce-cli containerd.io",
41     "sudo service docker start",
42     "sudo usermod -aG docker root",
43     "sudo curl -L \"https://github.com/docker/compose/releases/download
/1.25.3/docker-compose-$(uname -s)-$(uname -m)\" -o /usr/local/bin/docker-
compose",
44     "sudo chmod +x /usr/local/bin/docker-compose",
45
46     # Prodef setup and run
47
48     "ssh-keyscan github.com >> ~/.ssh/known_hosts",
49     "git clone -b deployment git@github.com:ULL-prodef/prodef-gui-frontend.
git",
50     "git clone -b deployment git@github.com:ULL-prodef/prodef-gui-backend.
git",
51     "git clone git@github.com:ULL-prodef/prodef-solver-rust.git",
52     "git clone git@github.com:ULL-prodef/prodef-docker-compose.git",
53     "cp /root/.npmrc ./prodef-gui-frontend",
54     "mv /root/.npmrc ./prodef-solver-rust",
55     "mv /root/prodef-backend.env ./prodef-docker-compose",
56     "cd prodef-docker-compose",
57     "docker-compose up -d --build",
58 ]
59 connection {
60     type = "ssh"
61     user = "root"
62     host = "${self.ipv4_address}"
63 }
64 }
65 }

```

Bibliografía

- [1] Avi. An introduction to terraform for beginners – terraform tutorial. <https://geekflare.com/terraform-for-beginners/>, 2022. (Visitado 06-07-2022).
- [2] Juan Pablo Caballero-Villalobos and Jorge Andrés Alvarado-Valencia. Greedy randomized adaptive search procedure (grasp): A valuable alternative for minimizing machine total weighted tardiness. http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S0123-21262010000200004, 2010. (Visitado 15-06-2022).
- [3] Wesley Chai and Kathleen Casey. Software as a service (saas). <https://www.techtarget.com/searchcloudcomputing/definition/Software-as-a-Service>, 2021. (Visitado 21-02-2022).
- [4] Baha Chammakhi. Dockerizing a react app with nginx, using multi-stage builds. <https://typeofnan.dev/how-to-serve-a-react-app-with-nginx-in-docker/>, 2020. (Visitado 21-06-2022).
- [5] Gara Miranda Coromoto León and Carlos Segura. Metco: A parallel plugin-based framework for multi-objective optimization. *International Journal on Artificial Intelligence Tools*, 18(4):569–558, 2009.
- [6] Daniel del Castillo de la Rosa. Prodef-algorithm: Interfaz para el modelado de meta-heurísticas. Technical report, Universidad de La Laguna, 2022.
- [7] GitHub Docs. Creating an oauth app. <https://docs.github.com/en/enterprise-server@3.4/developers/apps/building-oauth-apps/creating-an-oauth-app>, 2022. (Visitado 29-06-2022).
- [8] IBM Docs. Managing addresses and ports for http server. <https://www.ibm.com/docs/en/i/7.2?topic=tasks-managing-addresses-ports>, 2021. (Visitado 21-06-2022).
- [9] Juan J. Durillo and Antonio J. Nebro. jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10):760–771, 2011.
- [10] Neil Fraser. Ten things we’ve learned from blockly. In *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*, pages 49–50, 2015.
- [11] Andrés Calimero García Pérez. Prodef: meta-modelado de problemas de optimización combinatoria. Technical report, Universidad de La Laguna, 2020.
- [12] Daniel González Expósito. Prodef-gui: Interfaz gráfica para el modelado de problemas. Technical report, Universidad de La Laguna, 2021.

- [13] HashiCorp. Terraform documentation. <https://www.terraform.io/docs>, 2022. (Visitado 06-07-2022).
- [14] Graydon Hoare. The cargo book. <https://doc.rust-lang.org/cargo/commands/cargo-build.html>, 2022. (Visitado 16-06-2022).
- [15] ICANN. Internet corporation for assigned names and numbers. <https://www.icann.org/es>, 1998. (Visitado 29-06-2022).
- [16] DigitalOcean Inc. Digitalocean - the developer cloud. <https://www.digitalocean.com/>, 2011. (Visitado 07-07-2022).
- [17] Docker Inc. Docker development best practices. <https://docs.docker.com/develop/dev-best-practices/>, 2022. (Visitado 20-04-2022).
- [18] Docker Inc. Docker overview. <https://docs.docker.com/get-started/overview/>, 2022. (Visitado 23-02-2022).
- [19] Docker Inc. Networking overview. <https://docs.docker.com/network/>, 2022. (Visitado 28-06-2022).
- [20] Docker Inc. Overview of docker compose. <https://docs.docker.com/compose/>, 2022. (Visitado 21-04-2022).
- [21] MongoDB Inc. MongoDB documentation. <https://www.mongodb.com/docs/>, 2022. (Visitado 15-06-2022).
- [22] DATALOUDER (Adamo Jordan). Docker, la guía completa para aprender docker en 2021. <https://www.youtube.com/watch?v=n8R2LxdbtXQ>, 2020. (Visitado 21-02-2022).
- [23] Mwiza Kumwenda. The key to creating docker images: Dockerfile keywords. <https://betterprogramming.pub/the-key-to-creating-docker-images-dockerfile-keywords-57f25d7e5fa6>, 2022. (Visitado 19-04-2022).
- [24] Peter McKee. How to use the official nginx docker image. <https://www.docker.com/blog/how-to-use-the-official-nginx-docker-image/>, 2022. (Visitado 20-06-2022).
- [25] Judy Nduati. Docker security - best practices to secure a docker container. <https://www.section.io/engineering-education/best-practices-to-secure-a-docker-container/>, 2021. (Visitado 20-04-2022).
- [26] Miguel Angel Ordóñez Morales. Prodef: Diseño, implementación y experimentación con nuevos resolutores. Technical report, Universidad de La Laguna, 2022.
- [27] Herman J. Radtke III. Proxying api requests in development. <https://create-react-app.dev/docs/proxying-api-requests-in-development/>, note=(Visitado 21-06-2022), 2020.
- [28] React. Adding custom environment variables. <https://create-react-app.dev/docs/adding-custom-environment-variables/>, 2022. (Visitado 20-06-2022).

- [29] Igor Sysoev. Nginx docs. serving static content. <https://docs.nginx.com/nginx/admin-guide/web-server/serving-static-content/>, 2022. (Visitado 20-06-2022).
- [30] Matt Walters. Meta tool repository. <https://github.com/mateodelnorte/meta>, 2021. (Visitado 15-03-2022).
- [31] Stephen Watts. What is xaas? everything as a service explained. <https://www.bmc.com/blogs/xaas-everything-as-a-service/>, 2020. (Visitado 21-02-2022).