Facultad

de

Ciencias

EVALUACIÓN Y DESPLIEGUE DE HERRAMIENTAS DE GESTIÓN DE CONCURSOS DE PROGRAMACIÓN

(Evaluation and deployment of programming contests management tools)

Trabajo de Fin de Grado

para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Mario Fernández González

Director: Pablo Abad Fidalgo

Julio – 2022

Índice de Contenido

Índice de Figuras

Índice de Tablas

Resumen

La Consejería de Educación de Cantabria en colaboración con otras entidades y empresas organiza el Campeonato Regional de Informática “Bituca” para alumnos de educación primaria y educación secundaria (E.S.O y Bachillerato) durante el curso académico. La primera edición del campeonato se ha realizado en el año 2022 y debido al éxito de participación se espera que se realicen más ediciones del mismo.

Para la gestión de este tipo de campeonatos es habitual el uso de herramientas software de tipo CMS, un framework utilizado para la gestión de campeonatos de programación y facilitar las tareas administrativas asociadas a éstos, así como la evaluación de las soluciones propuestas.

Durante la primera edición se ha utilizado un software CMS llamado HackerRank, pero la herramienta es de pago por uso y volumen de usuarios de manera que parece recomendable encontrar una alternativa de código abierto que nos proporcione las funciones requeridas para la gestión del campeonato. Éste será el objetivo principal del TFG propuesto.

Para encontrar un sistema que se adapte a las necesidades del campeonato se analizará el tipo de herramientas disponibles, de las cuales se seleccionará un conjunto reducido para su evaluación. Se definirán un conjunto de requisitos para seleccionar el CMS más apropiado. Finalmente, se automatizarán todas las tareas de creación de concursos para facilitar al máximo la labor de despliegue de las pruebas.

Para cerrar el trabajo, se realizará un despliegue virtualizado del sistema, así como una prueba de carga de manera que se compruebe su correcto funcionamiento cuando hay un concurso activo y funcionando.

Abstract

The Regional Ministry of Education of Cantabria in collaboration with other entities and companies organises the Regional Computer Championship "Bituca" for students of primary and secondary education (E.S.O and Bachillerato) during the academic year. The first edition of the championship was held in 2022 and due to the success of participation it is expected that more editions will be held.

For the management of this type of contests it is common to use CMS software tools, a framework used for the management of programming contests and to facilitate the administrative tasks associated with them, as well as the evaluation of the proposed solutions.

During the first edition, a CMS software called HackerRank has been used, but the tool is pay-per-use and volume of users, so it seems advisable to find an open-source alternative that provides us the required functions for the management of the championship. This will be the main objective of the proposed TFG.

In order to find a system that suits the needs of the championship, the type of tools available will be analysed, from which a reduced set will be selected for evaluation. A set of requirements will be defined to select the most appropriate CMS. Finally, all contest creation tasks will be automated to make the deployment of the competitions as easy as possible.

To close the work, a virtualised deployment of the system will be performed, as well as a load test in order to check its correct functioning when a contest is active and running.

1 Introducción

Durante el curso académico 2021-22, la Consejería de Educación de Cantabria, en colaboración con la Universidad de Cantabria y diversas entidades (Ascentic, OpenWebinars) y empresas (CIC, Samsung, Microsoft), ha llevado a cabo el primer campeonato regional de informática "Bituca" [1]. Dicho campeonato es, en esencia, un concurso de programación enfocado a alumnos de educación primaria, E.S.O y bachillerato.

En su primera edición, el campeonato ha constado de varias fases. Entre los meses de Octubre y Febrero se arrancó en los centros educativos una fase preliminar de preparación. En esta fase se organizaron sesiones de formación y apoyo para comenzar a trabajar las destrezas y herramientas que correspondían a cada categoría. Se planificó una segunda fase clasificatoria, para llevar a cabo en el mes de marzo en caso de que el número de equipos inscritos excediera el máximo definido para la última fase del campeonato, realizando pruebas similares a las propuestas para la fase final.

Figura 1. Logotipo del campeonato "Bituca"

Tras dichas fases preparatorias, la fase final de la olimpiada tuvo lugar el día 5 de marzo en la Facultad de Ciencias de la Universidad de Cantabria. Divididos en 7 categorías nombradas por letras de la A hasta la F en orden alfabético atendiendo a su edad, a dicho evento acudió una numerosa representación de los centros de primaria y secundaria de la región, sumando un total de más de 250 participantes.

Dicha fase consistió en numerosas pruebas de diferente dificultad, en función de la categoría (curso escolar) en la que participen los alumnos. En las categorías inferiores, correspondientes a alumnos de tercero y cuarto de primaria, las pruebas consistieron en ejercicios de programación por bloques utilizando la aplicación Scratch, programación de robots utilizando un kit de Lego y, por último, diseño de circuitos a través del juego Minecraft para diseñar circuitos dentro del mismo. Mientras tanto, en la categoría F, que engloba a los alumnos de Bachillerato, se realizaron pruebas de programación en las que los usuarios resolvían los problemas planteados desarrollando programas escritos en C++, Python o Java.

Todos los ejercicios de estas pruebas comparten una estructura similar; se proporciona al alumno un enunciado que detalla el objetivo de la prueba y una descripción de los formatos de la entrada y salida esperados. A la descripción se añade en ocasiones un ejemplo concreto de entrada y su

correspondiente salida. En la Figura 2 se puede ver un enunciado de la categoría F extraído de la primera edición de la competición.

P05 Completando ADN

El ácido desoxirribonucleico, conocido también por las siglas ADN, es un ácido nucleico que contiene las instrucciones genéticas usadas en el desarrollo y funcionamiento de todos los organismos vivos y algunos virus. También es responsable de la transmisión hereditaria, siendo la función principal de la molécula de ADN el almacenamiento a largo plazo de información para construir otros componentes de las células.

En cadenas de ADN, los símbolos "A"(Adenina) y "T"(Timina) son complementos entre sí, como "C"(Citosina) y "G"(Guanina). Desarrollar un programa que recibe una cadena de ADN y devuelva su valor complementario.

Entrada

La entrada consiste en una secuencia de ADN representada por las letras A, C, G y T de forma consecutiva, sin espacios ni caracteres separadores.

Se establece que la cadena más larga no superará los 100 elementos.

Salida

Por cada caso de prueba se escribirá, en una línea independiente, la cadena complementaria.

Figura 2. Enunciado Bituca Categoría F

Tras desarrollar una solución, los alumnos envían su código para llevar a cabo un proceso automatizado de corrección. En dicho proceso se definen una serie de casos de prueba, con su solución esperada. Parte de dichos casos son públicos, pues se da la posibilidad al alumno de realizar múltiples envíos, y otra parte quedan ocultos para evitar códigos “maliciosos”. En base a estos casos de prueba, se corrige la solución asignando mayor o menor puntuación en función de cuantos funcionen correctamente.

Dado el éxito de la convocatoria, que hace previsible que el número de participantes crezca en los próximos cursos, la gestión de las pruebas requiere de herramientas específicas que permitan agilizar este proceso. En este tipo de concursos es habitual contar con herramientas que ayuden a la gestión de participantes, pruebas a realizar, evaluación de soluciones propuestas por los participantes y otras tareas administrativas. Estas herramientas son conocidas como CMS (Contest Management Systems por sus siglas en inglés) y permiten gestionar un concurso de programación en todos sus ámbitos.

El objetivo principal de un CMS es la definición de uno o varios concursos activos que constan de diferentes pruebas que los participantes deben superar. Estas pueden tener uno o varios casos de prueba sobre los ejecutar las soluciones y evaluarlas en función del número de casos superados correctamente, asignando mayor o menor puntuación a los usuarios. Por último, este tipo de herramientas suele contar con un ranking en tiempo real en el cual se pueden ver las puntuaciones de los usuarios detalladas en mayor o menor medida en función del sistema utilizado.

En la primera edición del campeonato la herramienta CMS utilizada ha sido el paquete software de nombre HackerRank [2]. Esta herramienta ha cubierto las necesidades de la primera edición del campeonato, ya que es una potente herramienta de evaluación de código de manera que la administración de la competición publicó las pruebas en el sistema.

Aunque el sistema nos permita realizar la evaluación del código, no nos permite usar otras funcionalidades como por ejemplo contar con un ranking en tiempo real con la evolución de la competición, la administración del campeonato era la encargada de recoger los datos de las evaluaciones del sistema y asignar las puntuaciones manualmente. Además, se trata de un software de pago por uso y número de usuarios, lo que puede suponer un aumento de costes en futuras ediciones. Estas limitaciones son las que han motivado el TFG realizado, cuyo objetivo principal ha consistido en la búsqueda, evaluación y despliegue de una herramienta CMS de tipo OpenSource capaz de cumplir con los requisitos del campeonato. Con dicho objetivo en mente, el resto del documento se ha organizado de la siguiente forma:

- En el Capítulo 2 se ha llevado a cabo una búsqueda exhaustiva de las herramientas de tipo CMS existentes en la actualidad. Se definen sus principales características y se hace un primer proceso de selección con el objetivo de trabajar con un grupo reducido de herramientas que mejor se adapte a los condicionantes de la prueba. Los sistemas elegidos han sido CMS [3] y DomJudge [4].
- En el Capítulo 3 se revisan los condicionantes para las pruebas, con el objetivo de hacer una especificación de requisitos que nos permita evaluar de manera precisa qué herramienta es la más adecuada para nuestro propósito. Los requisitos intentan cubrir aspectos tales como requerimientos hardware/software, facilidad de instalación y configuración o facilidad de uso y despliegue.
- El Capítulo 4 describe las tareas de despliegue en un entorno virtualizado. Este tipo de entornos dan una alta flexibilidad, que permitiría incluso despliegues en entornos de tipo Cloud. Se han trabajado dos enfoques de virtualización alternativos, uno basado en contenedores [5] y otro en máquinas virtuales [6].
- Una vez se cuenta con el sistema desplegado, en el Capítulo 5 se describirá la prueba de concepto realizada, la cual consistirá en la creación de un pequeño campeonato con sus correspondientes pruebas, usuarios y equipos de manera que se compruebe el correcto funcionamiento del sistema y sus distintas funcionalidades.
- Finalmente, cierran el documento los apartados de Conclusiones, la Bibliografía y los Anexos.

2 Estado del Arte

Desde sus primeras ediciones a principios de los años 90 [7], los campeonatos y olimpiadas de programación se han convertido en eventos habituales en todas las materias educativas relacionadas con la informática. Su número no ha dejado de crecer, llegando a realizarse en numerosos países, contando con fases regionales y nacionales, mediante las cuales se accede a una final a nivel internacional. En paralelo a su aumento de popularidad, han surgido múltiples sistemas de apoyo a la gestión de este tipo de eventos. Los objetivos principales de este tipo de paquetes software se pueden dividir en tres:

- Creación de los ejercicios de prueba, así como todos sus datos asociados: enunciado, soluciones, test de prueba, información sobre la forma de enviar soluciones.
- Configuración de la/las máquinas que tanto el CMS como los participantes utilizarán, haciendo especial hincapié en aspectos relacionados con la seguridad.
- Gestión de la celebración del evento: evaluar soluciones, proporcionar *feedback* a los participantes, monitorización de ranking de resultados, etc.

La cantidad de sistemas de tipo CMS disponibles en la actualidad es muy elevada. Se ha convertido en práctica habitual que las olimpiadas a nivel estatal de gran cantidad de países cuenten con su propia infraestructura de gestión para el evento [8] . En este capítulo solamente haremos mención a los CMS más populares a nivel internacional.

Uno de los CMS más populares es PC² CCS [9], cuya primera versión fue desarrollada para el concurso local de programación de la Universidad Estatal de California, en Sacramento, en el año 1990. Se trata de una herramienta escrita en Java, que opera con una arquitectura de tipo cliente-servidor que puede trabajar sobre infraestructuras heterogéneas (mezclando equipos Linux y Windows). Define 4 tipos de roles para interactuar con el concurso: Administrador, equipo, juez y marcador. La configuración de las pruebas y los roles se lleva a cabo a través de un entorno gráfico, con gran cantidad de parámetros disponibles. La herramienta es totalmente flexible en cuanto el lenguaje de programación escogido para las pruebas.

Otro de los CMS más utilizados es Mooshak [10], un sistema completo para la gestión de concursos desde la parte administrativa hasta la parte de evaluación y que fue desarrollado para concursos del tipo ICPC [11], aunque puede utilizarse y adaptarse para otro tipo de concursos. Dicha adaptabilidad ha hecho que este sistema pase a ser utilizado mayoritariamente por universidades como herramienta para evaluar código de prácticas de los alumnos.

Una alternativa que se desarrolló para sustituir a PC² CCS es RockTest [12], añadiendo alguna funcionalidad más que este, como por ejemplo permitir a los jueces un mayor control sobre las soluciones, pudiendo verlas y recalificarlas a través de la interfaz, permitir varios lenguajes de programación a la vez o permitir acceso a algunas de las funcionalidades de UNIX al código que se ejecuta. La herramienta está desarrollada en Python sobre la librería TkInter GUI y en 2003 se utilizó por primera vez en un pequeño concurso con 30 participantes a modo de prueba, de ahí en adelante, ha sido utilizado en numerosas ocasiones.

Como alternativa a los CMS podemos encontrar varios sistemas específicos para la corrección automática de tareas, utilizados en varias universidades alrededor del mundo como por ejemplo ECMS [13], un sistema diseñado para la elección de estudiantes por parte de las universidades. En este sistema los participantes deben realizar una o varias tareas y subirlas al sistema de manera que el sistema comprueba de manera automática en base a tests si las soluciones son correctas.

Existe una alternativa en España llamada Online Judge [14], este sistema lo mantiene la Universidad de Valladolid, aunque en la actualidad se encuentra algo obsoleto y, aunque se está desarrollando una nueva versión del sistema, esta se encuentra aún en fases tempranas de desarrollo. En lugar de permitir una instalación local, OnlineJudge se utiliza en la nube y se desarrolló en primer lugar para facilitar las labores de corrección de código de las prácticas de los alumnos, siendo liberado al público dos años más tarde, cuando alojó un concurso del tipo ICPC, más concretamente las fases regionales del suroeste de Europa, en el año 1999. Desde entonces, OnlineJudge cuenta con un total de 63.000 usuarios y 5,9 millones de soluciones presentadas.

En la actualidad, dos de las herramientas de gestión de concursos más extendidas son DOMJudge [4] y CMS [3]. Estos sistemas han sido utilizados en múltiples ocasiones en concursos del tipo ICPC [11] y en varias olimpiadas locales de informática en diversas partes del mundo. Son similares en funcionalidad y arquitectura al resto de herramientas descritas en los párrafos previos y cumplen, a priori, con todos los requisitos que impone la olimpiada regional. Gracias a su extendido uso, se trata de herramientas de fiabilidad contrastada, con un desarrollo activo en la actualidad y con proyección de futuro, por lo que serán nuestras candidatas como *framework* de gestión de concurso. En los siguientes apartados hacemos una descripción detallada de ambas herramientas

2.1 CMS

CMS se desarrolló inicialmente para las olimpiadas internacionales de informática que tuvieron lugar en septiembre de 2012 y desde entonces se ha utilizado en más ediciones de las mismas. La estructura de la herramienta está basada en varios servicios que se reparten las funcionalidades descritas en la introducción de este capítulo y una base de datos que gestiona toda la información de los concursos. Los servicios están programados en Python, escritos sobre una librería RPC denominada AsyncLibrary [15], esto es, una librería que sirve para creación de servidores y clientes, así como para abrir sockets de conexión asíncronos. Algunos de estos servicios contienen una segunda interfaz de conexión utilizando el *framework* web Tornado [16] para realizar la comunicación entre servicios. CMS utiliza PostgreSQL para implementar la base de datos del sistema. En la Tabla 1 se puede encontrar un resumen de los servicios, con una breve descripción de su funcionalidad y la capacidad de replicación que tiene cada uno de ellos, esto es, si el servicio puede tener varias instancias del mismo, para así proporcionar una mayor capacidad de cómputo, así como asegurar que no perdamos un servicio al perder una instancia.

Tabla 1. Servicios CMS

Nombre	Replicable	Función
LogService	No	Recibir, agregar y mostrar los logs del sistema.
Worker	Sí	Compilar y evaluar las soluciones en un entorno aislado.
EvaluationService	No	Mantener la cola de los trabajos para ser asignados a los *Worker*s.
ScoringService	No	Transformar los resultados de la evaluación en puntuaciones y añadirlas al ranking.
Checker	No	Realizar comprobaciones de actividad de todos los servicios.
ResourceService	Sí	Recoger información de los recursos de la máquina en la cual se encuentra el sistema e iniciar el resto de servicios de la máquina.
ContestWebServer	Sí	Proveer una página web a los participantes.
AdminWebServer	No	Proveer una página web a los administradores.

Debido a la necesidad de ejecutar cualquier código que creen los participantes, como este puede no ser seguro, es necesario realizar la compilación y ejecución en un entorno aislado, de manera

que, si por algún casual el código fallase al nivel de corromper la máquina en la que se ejecuta, no haya ningún problema con la máquina.

Para esto utiliza una sandbox, esto es un mecanismo de seguridad que proporciona un entorno aislado en el cual se asigna un espacio reservado en disco y solo pueden acceder a ese espacio, permite montar un sistema de archivos temporal para no tener que usar el del sistema operativo, tiene un espacio reservado en memoria y no puede acceder al resto, junto con otras muchas restricciones. Esto nos permite ejecutar cualquier cosa que necesitemos en un entorno virtual que no afecta al resto del sistema.

El sistema tiene un modelo de datos en el cual nos encontramos dos tipos: archivos y datos. Por archivos entendemos los ficheros en los que se encuentran los casos de prueba, las soluciones de los participantes y los resultados de compilarlas mientras que, por datos, entendemos aquello que no existe como un fichero, sino que son objetos de clases internas del sistema pero que también tenemos que almacenar. El almacenamiento del sistema está implementado sobre PostgreSQL, utilizando "Large Objects" para almacenar los archivos y usando SQLAlchemy para realizar el mapeado de objetos de Python a una base de datos de tipo SQL, almacenándolos también en PostgreSQL.

En la Figura 3 se muestra un diagrama con la arquitectura de la aplicación y la interconexión entre servicios. Cuando un usuario propone una solución a un problema lo hace a través del ContestWebServer y este, envía la solicitud de evaluación al EvaluationService, el cual se encarga de enviar la solución al *worker*. Una vez el *worker* tiene la solución, la compila y ejecuta, y devuelve los resutados de la ejecución al EvaluationService, tras esto, se solicita al ScoringService que, en función del resultado de las ejecuciones, asigne la puntuación correspondiente a la solución y la añada al ranking de la competición.

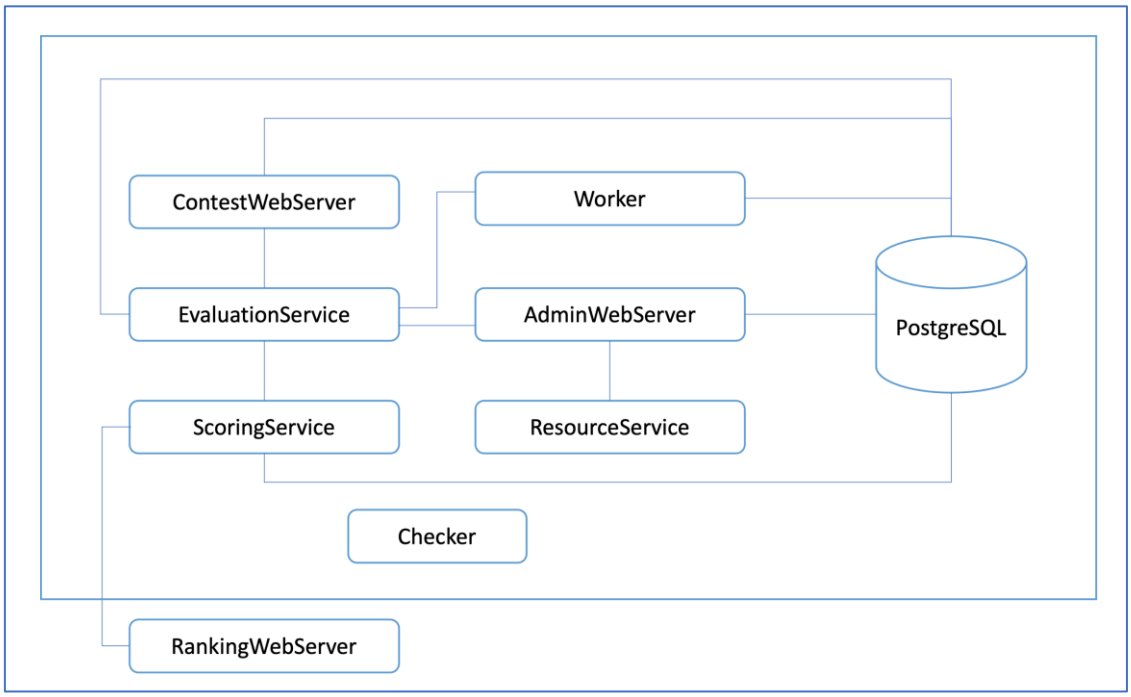

Figura 3. Arquitectura CMS

2.2 DOMJudge

DOMJudge fue utilizado por primera vez en el Benelux Algorithm Programming Contest en el año 2004 y ha seguido siendo utilizada desde esa fecha en varios concursos ICPC en algunas de las fases regionales y en la final mundial. También ha sido adaptada con éxito a niveles más pequeños para pequeños concursos a nivel de institutos.

La herramienta está principalmente programada en PHP, aunque cuenta con pequeñas partes en C y Shell Scripts. Está compuesta por varios servicios, los cuales se encargan de administrar las pruebas, mantener los servidores web, compilar y evaluar las soluciones. Dichos servicios son un Proxy, un servidor web de tipo Apache, una API REST y los *worker*s. En la Tabla 2 se resume la funcionalidad de los servicios mencionados.

Tabla 2. Servicios DOMJudge

Nombre	Replicable	Función
Proxy	No	Recibe las peticiones de los usuarios a las distintas partes del servidor web y se encarga de gestionarlas y enviarlas al lugar correcto.
WebServer	No	Servicio principal compuesto de tres subservicios, se encargada de proveer una interfaz a los distintos usuarios del sistema (administrador, participantes, publico)
PublicWebServer	No	Servicio que proporciona una interfaz web en la cual se puede visualizar el ranking en tiempo real del concurso de manera pública.
TeamsWebServer	No	Servicio que proporciona una interfaz web en la cual los participantes pueden ver los problemas planteados en el concurso y enviar las soluciones.
AdminWebServer	No	Servicio que proporciona una interfaz web en la cual los administradores del sistema pueden realizar todas las tareas correspondientes a su rol.
JudgeHost	Sí	Servicio encargado de compilar, evaluar y puntuar las soluciones propuestas por los participantes.
API REST	No	Servicio que sirve como punto de interconexión del resto de servicios del sistema.

De igual manera que CMS, DOMJudge necesita compilar y ejecutar las soluciones en un entorno aislado, pero, a diferencia de CMS, en este caso se utiliza una herramienta del *kernel* de Linux llamada cgroups. Esta herramienta nos permite agrupar y organizar los procesos del sistema de manera jerárquica, dando más importancia unos que a otros, así como establecer límites de recursos a cada grupo que hayamos creado.

Para la gestión y almacenamiento de la información la herramienta utiliza una base de datos relacional de tipo SQL y está preparada para utilizar indistintamente MySQL o MariaDB. En esta base de datos se almacena toda la información del concurso y está conectada a los distintos servicios del sistema. Además, la base de datos puede estar replicada en varias máquinas para evitar una posible pérdida de información.

Las conexiones entre los servicios se realizan mediante una API REST, la cual se basa en una serie de controladores y servicios que proporcionan unos *endpoints* (URLs en las que se encuentran los recursos de la API) sobre los cuales se pueden realizar ciertas operaciones vía HTTP para poder crear, borrar, obtener y modificar datos de manera que, por ejemplo, si se quiere

conectar un JudgeHost al sistema, utiliza la API REST para conectarse al servidor y se queda en modo espera y preparado. Cuando un participante sube una solución al sistema, este envía la petición al JudgeHost a través de la API para que realice la compilación y ejecución de la nueva solución propuesta.

Esta API REST proporciona también varios *endpoints* públicos los cuales se pueden utilizar para consultar información sobre el sistema y las competiciones del mismo, así como para crear nuevas competiciones y usuarios en el sistema.

En la Figura 4 se muestra un diagrama con las interconexiones de las distintas partes del sistema. Cuando un usuario tiene una nueva solución a un problema del campeonato, a través del TeamsWebServer la añade al sistema, el servidor se encarga a través de la API REST de enviar a un JudgeHost tanto la solución como la orden de compilar y ejecutar esa solución, en ese momento, cuando recibe la orden, compila, ejecuta y asigna una puntuación en función del número de casos de prueba superados correctamente. Finalmente realiza una petición a la API REST mediante la cual comunica el resultado de las ejecuciones y la puntuación al WebServer.

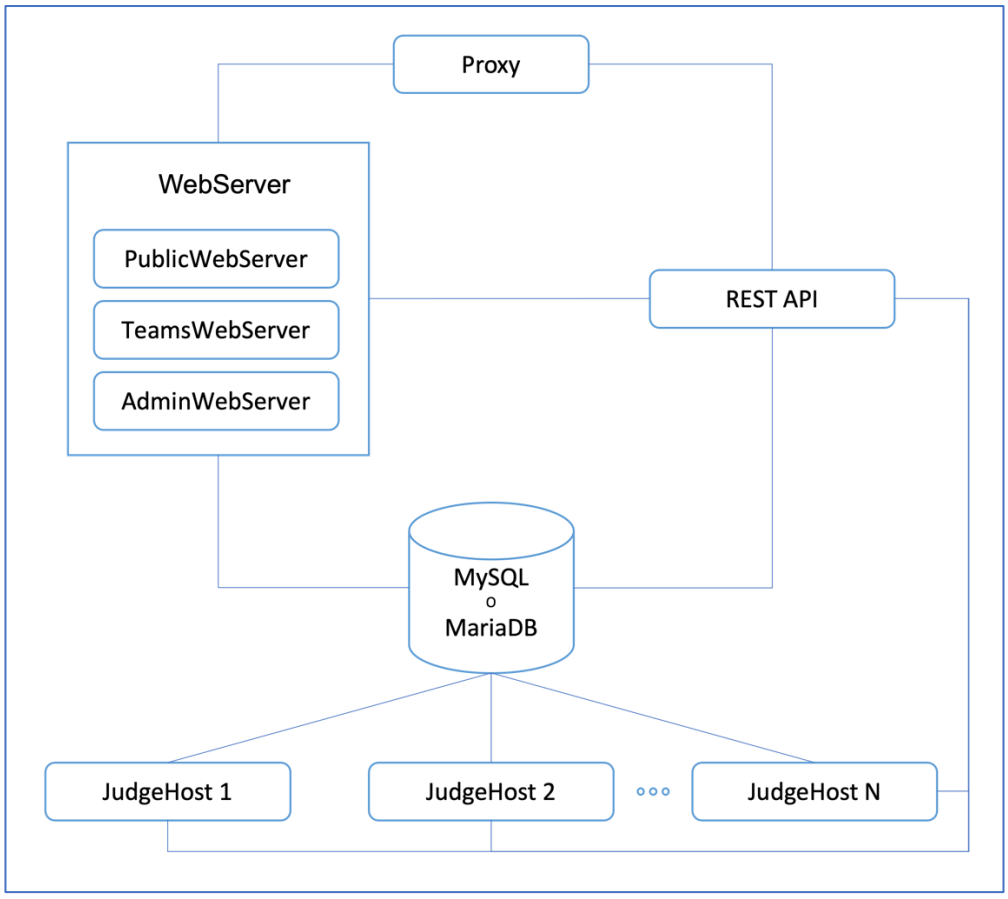

Figura 4. Arquitectura DOMJudge

3 Evaluación de Prestaciones

De cara a poder elegir el sistema más adecuado, es necesario evaluar ambas herramientas para ver si ambas cumplen con los requisitos de la competición que se quiere desarrollar. El objetivo es dar una estructura formal a los requerimientos que se describían en la propuesta de TFG y, a pesar de no ser un desarrollo software al uso contamos con algún requisito para el cual hay que realizar pequeños desarrollos por lo cual se ha optado por describir los requisitos con una especificación similar a la que se incluiría en los requisitos de un desarrollo de software. En esta sección se describen los requisitos y se realizará una evaluación de ambas herramientas en función a dichos requisitos para así determinar qué sistema es el que mejor se adapta a estos.

3.1 Especificación de Requisitos

Con el objetivo de evaluar ambos sistemas, es necesario realizar una especificación de requisitos en la cual se detallen las necesidades de los usuarios, así como las funciones necesarias del sistema para poder comprenderlas de manera correcta. Dichos requisitos se dividen en dos tipos: requisitos funcionales, los cuales se refieren a la funcionalidad del sistema y requisitos no funcionales, los cuales se refieren a características del sistema, a continuación, se detalla cada uno de ellos.

3.1.1 Requisitos Funcionales

Como se ha mencionado anteriormente, no contamos con un desarrollo de software al uso de manera que, los requisitos funcionales no son los que nos encontraríamos normalmente. Aun así dos de los requisitos de la competición no se refieren a ninguna característica del sistema a desplegar, sino que constan de dos automatizaciones a realizar con el sistema de manera que, se ha optado por incluirlos como requisitos funcionales. Estos se detallan en la Tabla 3.

Tabla 3. Requisitos Funcionales

Identificador	Nombre	Descripción
RF01	Automatización de despliegue	El usuario desea poder realizar un despliegue automatizado del sistema de manera que no se deba preocupar de los pasos de instalación y conexión de los componentes.
RF02	Automatización de tareas de importado de datos	El usuario desea poder realizar el importado de los datos de un concurso junto con sus participantes desde sistema de ficheros y, al ser posible, todos de golpe de manera que no sea necesario utilizar la interfaz administrativa para crear cada usuario y prueba.

3.1.2 Requisitos No Funcionales

Los requisitos no funcionales nos indican las características con las que debe de contar el sistema. Estos se encuentran detallados en la Tabla 4.

Tabla 4. Requisitos No Funcionales

Identificador	Nombre	Descripción
RNF01	Escalabilidad del sistema	Es necesario contar con un sistema escalable en función del número de participantes del campeonato. Se valorará la posibilidad de ampliar la capacidad del sistema en tiempo de ejecución.
RNF02	Sistema Open-Source	El conjunto del sistema debe de ser un conjunto de herramientas open source.
RNF03	Evaluación automática	El sistema debe de evaluar de manera automática con unos ajustes preestablecidos las soluciones propuestas por los participantes y asignar la puntuación correspondiente en base a esos ajustes.
RNF04	Diferentes categorías	El sistema debe de soportar tener varias categorías de manera que, en función de estas los usuarios participen en unas pruebas u otras.
RNF05	Diferentes lenguajes de programación	El sistema debe de soportar, al menos, los tres lenguajes de programación requeridos para la propuesta de soluciones: Python, Java y C.
RNF06	Contenido de las pruebas	Las pruebas constan de un enunciado, la explicación de la entrada y la salida esperada, así como varios ejemplos de entradas con sus correspondientes salidas. Los ejemplos de entradas y salidas deben de ser tanto públicos como privados, para tener alguno en cada prueba oculto a los participantes y así evitar que las soluciones simulen la salida en función de los ejemplos. Las pruebas creadas en los sistemas deben de poder contar con todos los datos.
RNF07	Datos del campeonato	El sistema debe de soportar añadir diversos datos al concurso que se esté gestionando tales como fecha y hora de inicio y fin, de manera que el campeonato comience y finalice automáticamente.
RNF08	Ranking de participantes	El sistema debe contar con un ranking con los participantes en función de su puntuación en tiempo real.
RNF09	Importado de datos	El sistema debe soportar la carga de los usuarios, equipos y pruebas de forma masiva.

3.2 Evaluación preliminar

De cara a evaluar las capacidades de ambos sistemas se han de tener en cuenta los requisitos no funcionales ya que, en base a estos, se decidirá cuál es el sistema que más se adapta a las necesidades del campeonato. A continuación, se comparan las distintas funcionalidades de ambos sistemas de manera que, al final, podamos tomar la decisión correcta.

3.2.1 Instalación de las herramientas

Previo a realizar cualquier evaluación necesitamos contar con las herramientas instaladas en un entorno de desarrollo, así como comprender los pasos y dificultad de instalación de cada una de las herramientas para tenerlo en cuenta a la hora de comparar y elegir una de ellas. Para esto se realiza una instalación de cada herramienta, en una máquina virtual, en modo *single node*, es decir, todos los componentes en la misma máquina.

Para esta instalación se utiliza como software de virtualización el gestor de máquinas virtuales VirtualBox [17] y como sistema operativo invitado Ubuntu 18.04 LTS, ya que es en el cual ambos sistemas han sido probados por parte de los desarrolladores.

Instalación de CMS

Para realizar el proceso de instalación para la herramienta CMS se han seguido los pasos indicados por el desarrollador, los cuales se pueden resumir en los siguientes puntos:

1. **Requisitos previos**: Instalación de las dependencias, disponibles en forma de paquetes, mediante el gestor de paquetes de Ubuntu, APT. Descarga del código sistema, el cual se encuentra disponible en la web oficial del proyecto [3] en formato tar.gz
2. **Creación de usuario**: para poder hacer uso del sistema, es necesario contar con un usuario el cual haga las veces de administrador. Este usuario debe ser definido a nivel de sistema operativo, así como un grupo dedicado para CMS [3].
3. **Instalación**: Una vez se han completado los pasos anteriores, hay varias opciones mediante las cuales podemos instalar el sistema: Instalación local, instalación virtualizada o instalación mediante gestores de paquetes tales como APT [18]. En este caso, se ha optado por realizar una instalación local. Para ello, dentro del directorio raíz de la descarga del sistema, se han instalado las dependencias necesarias de Python mediante Pip, el sistema de gestión de paquetes de Python. Tras esto, se arranca el proceso de instalación del servidor principal y de los *workers*.
4. Base de Datos: Antes de poder utilizar el sistema por primera vez, es necesario tener configurada la base de datos a utilizar, en este caso PostgreSQL. Para contar con una base de datos operativa se procede a su creación, se genera un usuario para CMS al que se otorga permisos de lectura y escritura y se inicializa la base de datos de acuerdo con los parámetros proporcionados por el sistema (proceso automatizado mediante un script SQL existente).

Una vez finalizado el proceso de instalación, se procede a realizar la configuración del *framework*. Para ello, el sistema cuenta con dos ficheros de configuración, uno para el servidor y otro para el ranking, una web en la que se publican las puntuaciones de los equipos en tiempo real según los participantes van solucionando los problemas. Es necesario editar los diferentes parámetros contenidos en ambos ficheros para permitir una correcta sincronización entre los diferentes servicios del sistema. Tras realizar la configuración del sistema, ya se puede poner en marcha mediante la ejecución de tres servicios: Servidor, logging y ranking web.

A continuación, una vez todos los servicios están funcionando de manera correcta, se puede comenzar a importar los datos al sistema (usuarios, equipos y pruebas a realizar). Esta tarea se puede realizar de forma manual (haciendo uso de la interfaz web) o automatizada a través de scripts. Debido al amplio volumen de datos con los que puede llegar a trabajar el concurso, carece de sentido añadir los datos uno a uno de forma manual. Para solucionar este problema, el sistema proporciona tres scripts para importar datos de manera masiva desde sistema de ficheros, uno para los usuarios, otro para las pruebas y otro para el concurso. Estos scripts están preparados para consumir los datos en tres formatos diferentes: YAML, Polygon y TPS (exclusivo para las pruebas) y añadirlos al sistema, persistiendo la información en la base de datos.

Durante el proceso de instalación y configuración del sistema se han tenido que solventar diversos problemas de diferente magnitud, el más significativo debido a un fallo de conexión entre el servicio del ranking de los usuarios y el servidor del CMS. Este fallo estaba causado por una mala

parametrización del ranking. Una vez se establecen los parámetros adecuados, el servicio funciona de manera correcta.

Instalación de DOMJudge

De la misma forma que con CMS, se han seguido los pasos de instalación de la documentación oficial del sistema, los cuales son:

1. **Requisitos previos**: instalación de las dependencias con el gestor de paquetes de Ubuntu, APT. De la misma manera que con CMS se descarga el sistema de su web oficial [4] en formato tar.gz.
2. **Base de datos**: se ha optado por utilizar MySQL como servidor de base de datos debido a que es para el cual hay documentación oficial de DOMJudge y, aunque se instala con el resto de las dependencias en el primer paso, hay que configurarla para que pueda funcionar de manera correcta y segura. Esto implica, entre otras cosas, la redefinición de la contraseña del usuario administrador de la base de datos, la administración de limitaciones para el acceso remoto del usuario administrador o la modificación de tablas que vienen por defecto en MySQL. El sistema proporciona un script mediante el cual se inicializa la base de datos, para este paso es necesario utilizar las credenciales definidas en el paso 2. Este script se encarga de crear el usuario administrador del sistema y almacena su contraseña dentro del directorio "/etc" de la ruta de instalación del servidor.
3. **Instalación**: el sistema proporciona una serie de scripts para realizar la instalación, los cuales se ejecutan a través de "recetas" de un Makefile. Tras completar la instalación, es necesario realizar la configuración del servidor web, en este caso DOMJudge utiliza el servidor Apache2. Para realizar este proceso el sistema proporciona una configuración por defecto la cual se copia a las carpetas de configuración de Apache correspondientes y, tras esto, se reinicia el servicio.

Una vez completados todos los pasos anteriores el servidor del sistema ya está configurado y funcionando, pero, al contrario que CMS, no quedan instalados los *worker*s por defecto, sino que hay que instalarlos aparte, para ello se siguen los siguientes pasos disponibles también en la documentación oficial de DOMJudge:

1. **Requisitos previos**: En primer lugar, hay que instalar las dependencias necesarias mediante el gestor de paquetes de Ubuntu, APT.
2. **Instalación**: Con la misma descarga utilizada para instalar el servidor anteriormente, instalaremos los *worker*s, que de ahora en adelante denominaremos Judgehosts. Seguimos los mismos pasos que para el servidor: Configuración de la ruta de instalación, definición e instalación de los Judgehosts apropiados.
3. **Seguridad**: por motivos de seguridad los Judgehosts se tienen que definir con un conjunto de permisos muy limitado. Para ello se crea un usuario y grupo específico, con un entorno de ejecución aislado a través de cgroups. La creación de los cgroups que necesita utilizar el sistema se lleva a cabo habilitando un servicio que proporciona el sistema a través de systemd.

Una vez tenemos instalados todos los componentes del sistema, aún no se puede utilizar, ya que es necesario configurar los Judgehosts para que se conecten con el servidor. Esta conexión se realiza a través de una API Rest, para la cual hay que configurar credenciales de acceso al servidor. Dentro de la ruta de instalación del servidor en el directorio "/etc" contamos con un fichero en el cual tenemos las credenciales para conectar los Judgehosts al servidor, este fichero

se debe de copiar al mismo directorio en la ruta de instalación de cada Judgehost, de manera que la utilicen en el inicio de la conexión.

Una vez tenemos la configuración de la API Rest realizada, hay que inicial los Judgehosts que se hayan instalado, para ello el sistema ofrece dos posibilidades, la primera de ellas consta de ejecutar un script que se encuentra dentro de la instalación del Judgehost, la segunda y más recomendable consiste en habilitar un servicio de systemd que ofrece el sistema de manera que el Judgehost se inicia junto con el inicio del sistema operativo.

Con todos los servicios del sistema funcionando ya se puede comenzar a importar los datos de usuarios, equipos, concursos y pruebas de los mismos. Para ello el sistema proporciona una interfaz web de administración en la cual se pueden realizar tareas de creación de usuarios, equipos, concursos y pruebas. Al igual que en CMS los datos que se añaden a través de la interfaz se tienen que añadir de uno en uno.

El sistema contempla la situación en la que se deseen añadir una gran cantidad de datos como, por ejemplo, importar todos los usuarios y datos de un concurso. El sistema proporciona una API REST mediante la cual podemos realizar peticiones HTTP de tipo POST para la creación de los diferentes datos posibles. Estas peticiones llevan en el cuerpo del mensaje los datos a importar, los cuales deben de ser en formato TSV a excepción de los concursos, que utilizan formato YAML y las pruebas, que se utiliza un archivo comprimido en formato ZIP.

El proceso de instalación de DOMJudge ha resultado más tedioso que el de CMS. Durante la instalación se han encontrado problemas cuya resolución ha requerido de un esfuerzo significativo, debido principalmente a una documentación deficiente en cuanto al proceso de instalación. Los principales problemas han sido los siguientes:

1. Las dependencias proporcionadas en la documentación para la parte de instalación del servidor no son correctas. Faltan 5 dependencias sin las cuales ni siquiera se puede llegar a instalar el sistema. Éstas son: libcgroup-dev, gcc, g++, libcurl4-openssl-dev, libjsoncpp-dev y make.
2. En la instalación del servidor, en el paso de configuración del servidor Apache, la documentación indica, entre otras cosas, cómo crear los enlaces simbólicos a los ficheros de configuración por defecto proporcionados por DOMJudge para el servidor Apache. Esta parte del procedimiento genera problemas a la hora de reiniciar el servicio del servidor Apache, y la solución pasa por copiar los archivos en lugar de crear enlaces simbólicos.
3. De nuevo, la lista de dependencias proporcionadas en la documentación para la parte de instalación del Judgehost no está completa. Falta una dependencia requerida por el Judgehost para llevar a cabo el proceso de compilación y ejecución de una solución propuesta a una de las pruebas por un usuario. Esta dependencia es unzip.

3.2.2 Comparación de las herramientas

A mayor número de requisitos cumplidos, mejor se adaptará la herramienta a las necesidades del concurso tal y como está planteado en la actualidad. En esta sección se evalúa cada requisito no funcional para cada uno de los sistemas analizados.

RNF01 – Escalabilidad del sistema.

La arquitectura por defecto de CMS nos permite tener todos los *worker*s instalados en la misma máquina que el servidor principal, de manera que, si añadimos más *worker*s, al estar todos en la misma máquina, no ganamos rendimiento, quizás incluso podemos perder rendimiento por ejemplo debido a que un proceso puede bloquear a otro. En cambio, la arquitectura de DOMJudge nos permite contar con una máquina para cada servicio, dedicando tantas máquinas como se necesiten para los nodos de tipo *worker*. De esta manera, podemos aumentar el rendimiento aumentando el número de *worker*s, además, podemos añadir más *worker*s en cualquier momento, incluso si hay un concurso activo. Esto nos permite, por ejemplo, si un *worker* falla y se queda irrecuperable, poder arrancar otro más de manera sencilla.

RNF02 – Sistema open-source.

Ambas herramientas, así como sus dependencias y el sistema sobre el cual se ejecutan son de código abierto de manera que en este sentido ambos sistemas cumplen el requisito.

RNF03 – Evaluación automática

Ambas herramientas soportan la evaluación automática del código propuesto por los participantes en las soluciones. Esto incluye asignar puntuaciones automáticamente en función de una configuración preestablecida.

RNF04 – Diferentes categorías

CMS no nos permite asignar usuarios a ninguna categoría, solo a un equipo, mientras que DOMJudge, nos permite asignar un usuario tanto a un equipo como a una categoría. Aunque se pueda añadir un usuario a una categoría, el sistema no permite que dentro de un mismo campeonato haya unas preguntas asignadas a unas categorías y otras preguntas a otras categorías. Esto, en el caso de DOMJudge tiene una solución, ya que el sistema nos permite tener activos varios campeonatos a la vez, podríamos tener uno por categoría.

RNF05 – Diferentes lenguajes de programación

Ambos sistemas nos permiten utilizar los tres lenguajes de programación solicitados, así como otros lenguajes pudiendo incluso, en el caso de DOMJudge, adaptar el sistema para poder utilizar un lenguaje que no esté oficialmente soportado.

RNF06 – Contenido de las pruebas

En el caso de ambos sistemas, podemos contar con un fichero a modo de enunciado en formato PDF, entre otros. En este enunciado se pueden añadir tanto la descripción de la prueba como la descripción de entrada y salida esperados. Además, ambos sistemas nos permiten añadir múltiples casos de prueba para una prueba, pudiendo seleccionar si se desea que estos sean de carácter público o privado.

RNF07 – Datos del campeonato

Ambos sistemas nos permiten seleccionar una fecha y hora de inicio y fin de manera que una vez definido el concurso el sistema le publicará cuando sea el inicio y comenzará a admitir participaciones por parte de los usuarios finalizando estas de manera automática cuando esté definido el fin del concurso.

RNF08 – Ranking de participantes

Ambos sistemas cuentan con un ranking en tiempo real en el cual se pueden observar los puestos y puntos de los participantes de la prueba, aunque en el caso de DOMJudge el ranking viene integrado con el resto del sistema de manera que no es necesario configurarle, mientras que en CMS se trata de un servicio independiente que hay que arrancar y configurar para que se conecte al servidor principal.

RNF09 – Importado de datos

Ambos sistemas proporcionan una implementación para realizar la carga de datos de usuarios equipos y pruebas de manera masiva, aunque son dos aproximaciones bastante diferentes. En el caso de CMS, el sistema incorpora scripts mediante los cuales, desde archivos en el sistema de ficheros, podemos realizar la carga de los usuarios, equipos y pruebas al sistema, mientras que, en el caso de DOMJudge, el sistema nos proporciona una API Rest a la cual se pueden realizar peticiones HTTP de tipo POST enviando los datos a importar al *endpoint* correspondiente en la API podemos realizar la tarea de importado de datos. La opción de DOMJudge nos permite más flexibilidad ya que, en primer lugar, se puede realizar la petición desde cualquier lugar que tenga conexión con el servidor, mientras que en CMS se lanzan los scripts dentro del servidor, y permite realizar una gestión de errores a la hora de importar los datos gracias a los códigos de respuesta HTTP.

Documentación de las herramientas

En lo que respecta a la documentación de ambos sistemas, si bien CMS describe con algo más de detalle el proceso de instalación del sistema, la documentación de DOMJudge nos proporciona un mayor nivel de precisión en lo que se refiere a la gestión de los concursos, estructura de datos exacta para realizar la importación de los datos y utilización de la API, así como para solución de problemas que puedan existir durante la realización de las pruebas.

3.3 Conclusiones

Como podemos observar al analizar ambos sistemas, ambos son muy similares ya que han sido desarrollados con la misma idea y para realizar las mismas funciones en un concurso de programación del mismo tipo, ICPC. Tras la evaluación y comparación de ambos sistemas podemos ver qué sistema se adapta más a lo que necesitamos en base a los requisitos. En la Tabla 5 podemos ver un resumen del cumplimiento de los requisitos por parte de ambas herramientas.

Tabla 5. Comparativa ambos sistemas

Identificador	Nombre	CMS	DOMJudge
RNF01	Despliegue en entorno virtualizado	Sí	Sí
RNF02	Escalabilidad del sistema	Sí (tras configurar)	Sí
RNF03	Sistema Open-Source	Sí	Sí
RNF04	Evaluación automática	Sí	Sí
RNF05	Diferentes categorías	No	Sí
RNF06	Diferentes lenguajes de programación	Sí	Sí
RNF07	Contenido de las pruebas	Sí	Sí
RNF08	Datos del campeonato	Sí	Sí
RNF09	Ranking de participantes	Sí	Sí
RNF10	Importado de datos	Sí	Sí

Como podemos observar, DOMJudge cumple con todos los requisitos del campeonato mientras que CMS no cumple dos de ellos, al menos de manera directa, sin realizar tareas de configuración previa, de manera que, sin que sea necesario siquiera tener en cuenta los casos en los que ambos sistemas cumplen con el requisito y que DOMJudge se adapta mejor al caso concreto, el sistema elegido es DOMJudge ya que con CMS no se podría implementar el campeonato tal y como se desea. Aunque el sistema elegido sobre el que realizar las automatizaciones sea DOMJudge, ambos cumplen con prácticamente todos los requisitos y estas automatizaciones se podrían realizar de igual manera con CMS.

4 Despliegue del sistema

Tras concluir qué herramienta se adapta más a los requisitos definidos, se procede a realizar un despliegue "real" en un entorno virtualizado de la misma, en varias máquinas de manera distribuida de tal forma que los servicios del sistema se ejecuten en un entorno acotado (una máquina virtual o un contenedor en nuestro caso).

Como se definió en la sección 3.1.1, los requisitos funcionales del sistema requieren dos procesos de automatización. El primero está relacionado con el despliegue del sistema, con el objetivo de hacer una instalación personalizada de las máquinas necesarias (de acuerdo con su rol dentro del *framework*) y gestionar el proceso de arranque. El segundo tiene por objetivo automatizar los procesos de importado de datos al sistema como pueden ser los usuarios, equipos, categorías y pruebas de la competición. A continuación, se detalla el proceso seguido para realizar ambas automatizaciones.

4.1 Automatización de despliegue virtualizado

El primer requisito funcional consiste en la realización de un despliegue automatizado en un entorno virtualizado (RF01). Para cumplir con este requisito se realizan dos propuestas, cada una con un punto de vista algo diferente a través de contenedores o de máquinas virtuales. Un contenedor es una herramienta que se utiliza comúnmente para desplegar desde microservicios hasta aplicaciones de mayor tamaño. Se trata de un paquete de software que incluye todo lo necesario para ejecutar una aplicación: código, herramientas de sistema, librerías y parámetros de configuración. Los contenedores comparten el SO de la máquina en la que se ejecutan, por lo que no requieren una réplica del sistema operativo por aplicación. Por esta razón son muy ligeros y sobrecargan el sistema significativamente menos. Existen herramientas de orquestación para generar infraestructuras de cómputo, red y almacenamiento a través de múltiples contenedores como Kubernetes [19]. En este caso se opta por utilizar el gestor de contenedores Docker [20] ya que es el más utilizado en cuanto a desarrollo y para el que están preparadas las imágenes de DOMJudge.

A diferencia de un contenedor, una máquina virtual emula completamente a una máquina física, de manera que cuenta con un sistema operativo completo. Los entornos virtualizados se gestionan a través de un hipervisor, que puede ser de dos tipos. En los entornos virtualizados con un hipervisor de tipo 1, el hipervisor tiene acceso directo al hardware de la máquina, y no es necesaria la presencia de un Sistema Operativo para ejecutarse. En el caso de que el hipervisor no acceda al hardware directamente sino a través del sistema operativo contamos con un hipervisor de tipo 2. En este caso se opta por utilizar un hipervisor de tipo 1, más concretamente Hyper-V [21], y también se utiliza un software de orquestación de máquinas virtuales llamado Vagrant [22], el cual nos permite crear y configurar máquinas de una manera muy sencilla y directa utilizando imágenes pre montadas de diversos sistemas operativos.

La principal diferencia que encontramos entre contenedores y máquinas virtuales reside en que las máquinas virtuales, deben de tener obligatoriamente cada una un sistema operativo, de manera que si se necesitan desplegar cinco servicios, cada uno en una máquina, se necesitan cinco sistemas operativos mientras que, con contenedores, esos mismos cinco servicios solo necesitan una única máquina con un sistema operativo, sobre la cual se despliegan esos contenedores de manera que esta es una muy buena opción en entornos en los que no se cuenten con demasiados recursos, como por ejemplo, un entorno de desarrollo.

Otra diferencia es que los contenedores están preparados para iniciarse directamente, sin necesidad de tener que realizar ningún paso de instalación que sí necesitaríamos en el caso de una máquina virtual de manera que los contenedores son una manera más rápida de desplegar una aplicación o servicios.

4.1.1 Despliegue en Contenedores

Como se ha mencionado anteriormente para realizar el despliegue del sistema en contenedores se ha optado por utilizar Docker [20], junto con una de sus herramientas llamada Docker Compose [23], esta nos permite, utilizando un fichero de configuración, configurar y desplegar varios contenedores a la vez. DOMJudge cuenta con contenedores de Docker preparados para realizar el despliegue con esta herramienta. Para dicho despliegue se proporcionan dos contenedores: el primero de ellos, llamado DOMServer, se encarga de los servicios del servidor web, proxy y de la API REST, mientras que el segundo, llamado JudgeHost, se encarga del servicio homónimo. De esta manera se plantea realizar un despliegue de contenedores de manera que contaremos con la arquitectura de la Figura 5. Como se puede ver en la figura, además de los contenedores que proporciona DOMJudge, es necesario contar con un contenedor para la base de datos, en este caso, se utilizará el contenedor oficial de Docker para MySQL.

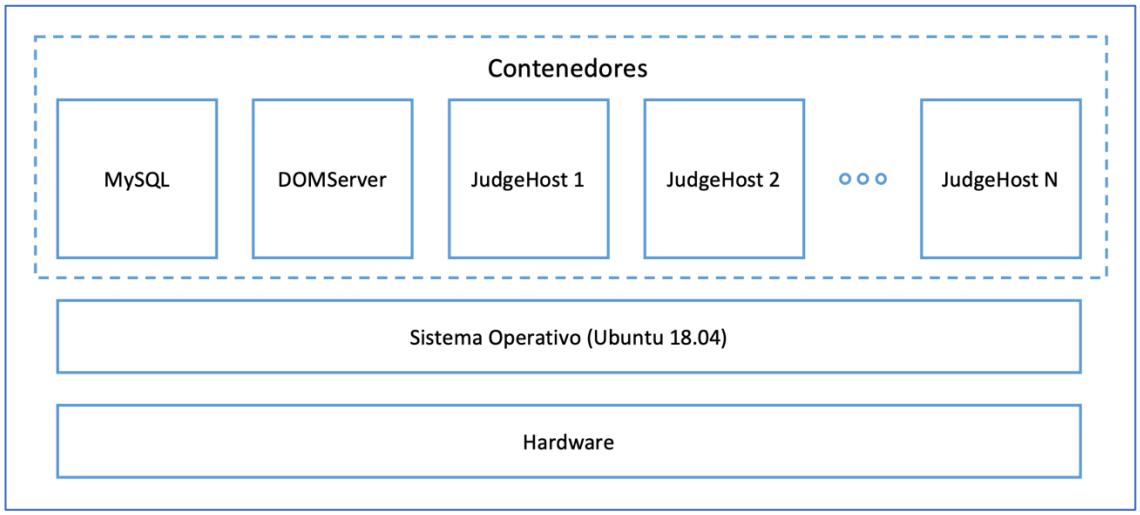

Figura 5. Diagrama de Despliegue en Contenedores

Para realizar el despliegue, en primer lugar, necesitamos ver las dependencias de unos contenedores con otros. En el caso del contenedor DOMServer, se necesita un contenedor de MySQL o MariaDB en el cual exista un usuario llamado domjudge, así como una base de datos homónima. Para el caso de los contenedores JudgeHost, es necesario tener un contenedor DOMServer al cual conectarse, así como la autorización para utilizar la API REST del DOMServer, la cual se genera automáticamente cuando se crea el DOMServer y tener habilitado los cgroups en el sistema operativo desde el cual se crean los contenedores, ya que el contenedor utiliza los cgroups de la máquina anfitrión.

El contenedor de MySQL nos proporciona una manera sencilla de realizar la configuración de una base de datos junto con un usuario asignado a la misma, esto es, utilizando variables de entorno en el contenedor en el momento del arranque de manera que configurando las variables *MYSQL_ROOT_PASSWORD, MYSQL_USER, MYSQL_PASSWORD y MYSQL_DATABASE*, se configura la base de datos indicada y se asignan los permisos del usuario indicado a esa base de datos.

Tras esto, podemos configurar el contenedor DOMServer de la misma manera que el de MySQL, utilizando variables de entorno para decirle al contenedor a que host de MySQL se debe conectar y que base de datos y credenciales debe utilizar. Las variables de entorno a configurar son *MYSQL_ROOT_PASSWORD, MYSQL_USER, MYSQL_PASSWORD, MYSQL_DATABASE y MYSQL_HOST*.

Además de las configuraciones ya indicadas, se indica también la redirección de puertos en ambos contenedores para poder usar la base de datos, así como el servidor web en el caso de DOMServer. Tras esto ya se pueden ejecutar los contenedores mediante docker compose y acceder al servidor web en la dirección localhost:3000, En la Figura 6 se proporciona un ejemplo de fichero de configuración para Docker Compose.

```
version: "3.9"
services:
  mysql:
    image: docker.io/mysql
    hostname: mysql
    ports:
      - "3306:3306"
    networks:
      dom_net:
        ipv4_address: 10.1.0.2
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
      interval: 10s
      timeout: 5s
      retries: 5
      start_period: 10s
    environment:
      - MYSQL_ROOT_PASSWORD=domjudge
      - MYSQL_USER=domjudge
      - MYSQL_PASSWORD=djpw
      - MYSQL_DATABASE=domjudge

  domserver:
    image: docker.io/domjudge/domserver:7.3.4
    hostname: domserver
    ports:
      - "3000:80"
    networks:
      dom_net:
        ipv4_address: 10.1.0.3
    environment:
      - MYSQL_ROOT_PASSWORD=domjudge
      - MYSQL_USER=domjudge
      - MYSQL_PASSWORD=djpw
      - MYSQL_DATABASE=domjudge
      - MYSQL_HOST=mysql
    depends_on:
      mysql:
        condition: service_healthy

networks:
  dom_net:
    driver: bridge
    ipam:
      config:
        - subnet: 10.1.0.0/24
```

Figura 6. Ejemplo de configuración para Docker Compose

Tras esto, ya podemos configurar el JudgeHost, utilizando las credenciales de la API REST creadas en la inicialización del contenedor DOMServer, utilizando también una variable de entorno. Debido a que para utilizar la API para conectar los JudgeHost al servidor necesitamos una contraseña generada automáticamente en el momento de la creación del mismo, no podemos

incluir en el fichero de docker compose el JudgeHost para arrancarle junto con los otros dos contenedores.

Ya configurados los contenedores para la base de datos y el DOMServer, necesitamos obtener las credenciales, esto lo podemos realizar ejecutando el comando *docker exec* al cual se le pasa como argumento un contenedor y que comando se desea ejecutar dentro del contenedor de manera que, haciendo un *cat* al fichero correspondiente (*INSTALL_DIR/etc/dbpasswords.secret*) tenemos las credenciales. De esta misma manera podemos obtener la contraseña inicial del administrador, haciendo un cat al fichero *INSTALL_DIR/etc/initialadminpassword.secret*

Tras esto, antes de poder crear los contenedores para los JudgeHost, debemos habilitar los cgroups, esto lo hacemos editando el fichero de configuración */sys/fs/cgroup,* añadiendo los siguientes parámetros de configuración a la variable *GRUB_CMDLINE_LINUX_DEFAULT: cgroup_enable=memory swapaccount=1 systemd.unified_cgroup_hierarchy=0*

Una vez habilitados los cgroups y habiendo obtenido las credenciales para la conexión con la API REST, ya podemos crear el contenedor o contenedores JudgeHost, pasando las credenciales de conexión en la variable de entorno `JUDGEDAEMON_PASSWORD`, compartiendo la carpeta del sistema operativo en la que se encuentran los cgroups, así como indicándole un id único al JudgeHost en la variable de entorno `DAEMON_ID`, asignando la misma red creada para los otros contenedores y asignando un nombre al contenedor. Todo esto lo podemos realizar de la misma manera que el resto de contenedores, con docker compose, como se muestra en la Figura 7, o con un comando de Docker, como se muestra en la Figura 8:

```
judgehost:
  image: docker.io/domjudge/judgehost:7.3.4
  hostname: judgehost
  networks:
    dom_net:
      ipv4_address: 10.1.0.4
  environment:
    - JUDGEDAEMON_PASSWORD=<Contraseña API REST>
    - DAEMON_ID=<ID JudgeHost>
  volumes:
    - /sys/fs/cgroup:/sys/fs/cgroup
  depends_on:
    - domserver
```

Figura 7. Ejemplo de configuración de JudgeHost para Docker Compose

```
docker run -it -d --restart=always --privileged -v /sys/fs/cgroup:/sys/fs/cgroup:ro -
e JUDGEDAEMON_PASSWORD=$rest --name judgehost-0 --network dom_net --hostname
judgedaemon-0 -e DAEMON_ID=0 domjudge/judgehost:7.3.4
```

Figura 8. Ejemplo de comando de Docker para iniciar un JudgeHost

Dado que para el contenedor JudgeHost necesitamos archivos del contenedor DOMServer y no podemos incluir todos los contenedores en el fichero de configuración de Docker Compose para iniciarlos al mismo tiempo, se ha desarrollado un pequeño script que se encarga de iniciar los contenedores de MySQL y DOMServer con Docker Compose, obtener las credenciales de DOMServer, preguntar por cuantos contenedores JudgeHost se desean iniciar e, iniciarlos, utilizando el ejemplo del comando de Docker. En la Figura 9 se muestra dicho script.

```
echo "Introducir número de judgehosts deseados:"
read num
echo "\nArrancando servidor..."
docker-compose up -d
echo "Waiting for server..."
sleep 30
echo "\nArrancando judgehosts..."

root_pass=$(docker exec -u root vagrant_domserver_1 cat
/opt/domjudge/domserver/etc/initial_admin_password.secret)
echo "Initial admin password for server: $root_pass"
rest=$(docker exec -u root vagrant_domserver_1 cat
/opt/domjudge/domserver/etc/restapi.secret | awk 'NR == 3 {print $4}')
i=0
while [ $i -lt $num ];do
    docker run -it -d --restart=always --privileged -v
/sys/fs/cgroup:/sys/fs/cgroup:ro -e JUDGEDAEMON_PASSWORD=$rest --name judgehost-$i --
network vagrant_dom_net --hostname judgedaemon-$i -e DAEMON_ID=$i
domjudge/judgehost:7.3.4
    i=$(($i+1))
done
```

Figura 9. Script de despliegue del sistema en contenedores

4.1.2 Despliegue en Máquinas Virtuales

La opción alternativa de despliegue consiste en utilizar máquinas virtuales, en este caso se ha utilizado el hipervisor de tipo 1 Hyper-V, junto con la herramienta de orquestación de máquinas virtuales Vagrant. Los hipervisores de tipo 1 son aquellos que se ejecutan directamente sobre el hardware de la máquina física en la que se utilizan, proporcionando mayor rendimiento que los hipervisores de tipo 2, los cuales necesitan ejecutarse sobre un sistema operativo, utilizando software que simula el hardware de la máquina física. En la Figura 10 se muestra un diagrama con las diferencias entre ambos tipos de hipervisores.

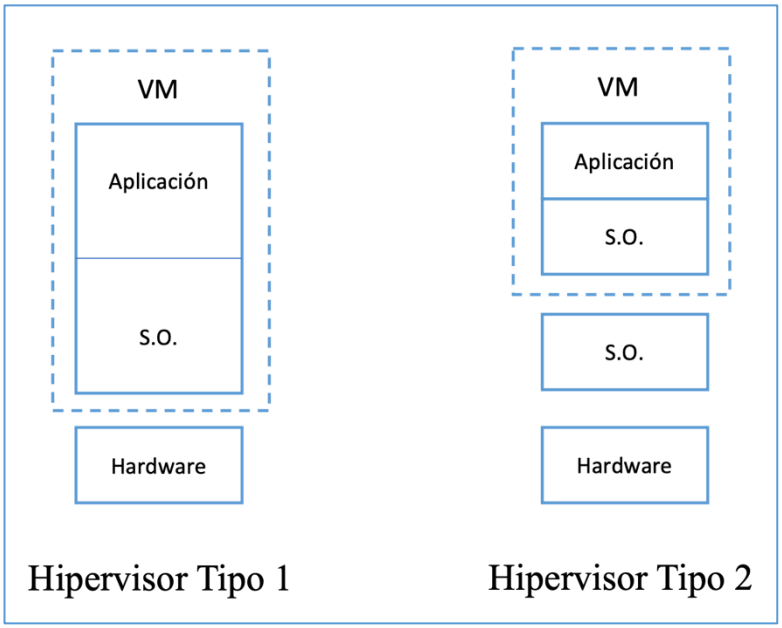

Figura 10. Tipos de Hipervisores

Además de un hipervisor es necesario utilizar un gestor de máquinas virtuales que tenga soporte para el hipervisor seleccionado, Vagrant en nuestro caso. A través de un fichero de configuración

y una imagen base de una máquina virtual suministrada a través de Vagrant Cloud, podemos iniciar una máquina virtual sin tener que realizar todo el proceso de instalación de la misma. Adicionalmente, Vagrant nos permite realizar aprovisionamiento de máquinas virtuales utilizando herramientas como puede ser Ansible [24], o scripts de Shell.

Para un despliegue virtualizado, el primer paso consiste en definir los servicios que tenemos que desplegar y en qué máquinas (de una forma similar al trabajo realizado en el apartado 4.1.1). De esta manera, al igual que en el despliegue por contenedores, tendremos una máquina que se encarga de la base de datos, en este caso MySQL, otra que se encarga del servidor DOMServer y otras n-máquinas que se encargan de los *worker*s JudgeHost. En la Figura 11 se muestra un diagrama con el despliegue del sistema. Para realizar el despliegue, hay que realizarlo, al igual que en el caso de los contenedores, en tres pasos: máquina con la base de datos, máquina con DOMServer y para finalizar máquina o máquinas con JudgeHost. Para crear las diferentes máquinas virtuales se ha seguido el proceso detallado a continuación.

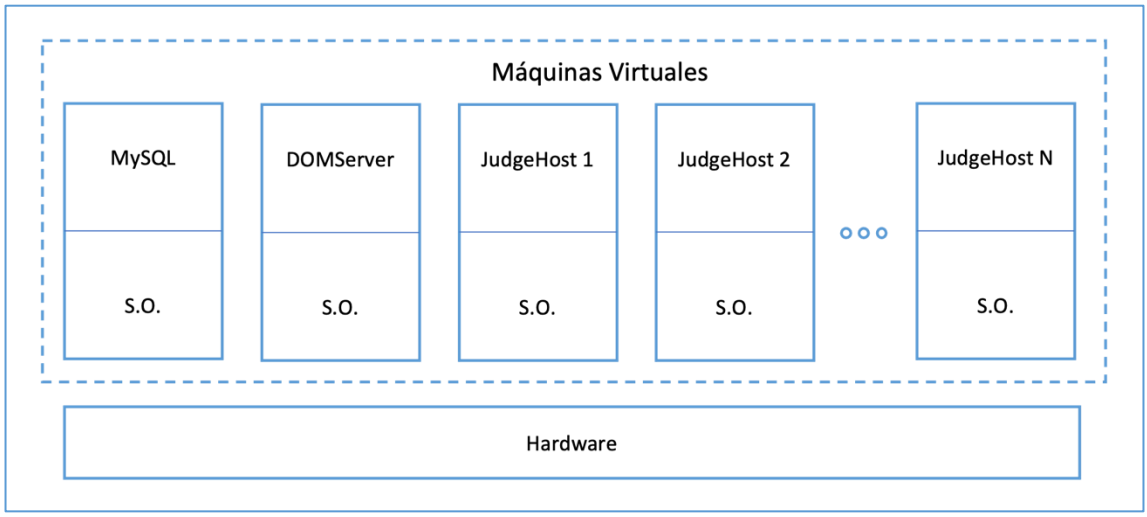

Figura 11. Diagrama de Despliegue en Máquinas Virtuales

En primer lugar, tenemos que elegir la imagen base sobre la que montaremos las diferentes máquinas de nuestro sistema. En este caso, se ha elegido una imagen con sistema operativo Ubuntu versión 18.04 LTS proporcionada por los desarrolladores de Vagrant. A partir de esta imagen podemos crear las distintas máquinas virtuales, a través de un fichero llamado Vagrantfile, en el cual se definen parámetros de las máquinas virtuales tales como el número de CPUs, la memoria, el tamaño de disco, la configuración de red y el aprovisionamiento de las máquinas virtuales. En este caso se ha optado por una configuración sencilla para probar los despliegues, asignando dos CPUs a cada máquina, 2 GB de memoria RAM y utilizando una red privada en la que se encuentren las máquinas para comunicarse entre ellas. Un ejemplo del fichero Vagrantfile con esta configuración se encuentra en el Anexo, Figura 15.

Sobre las diferentes máquinas debemos de realizar una serie de tareas de aprovisionamiento de cara a completar cada máquina con su software correspondiente. En primer lugar, nos tenemos que preocupar de la base de datos. Para realizar el aprovisionamiento de esta máquina se utiliza Ansible, una herramienta de aprovisionamiento desarrollada por Red Hat [25] y que, con un simple fichero con formato yml, nos permite describir distintas tareas a ejecutar en la máquina. En este fichero indicamos qué paquetes queremos instalar, en este caso mysql-server, así como otras tareas, como puede ser el cambio de la contraseña del usuario administrador de la base de

datos, la creación de la base de datos y el usuario para DOMJudge, o permitir a mysql escuchar peticiones remotas. Este fichero de configuración se encuentra en el Anexo, Figura 16.

Tras contar con la máquina de MySQL, debemos montar la máquina de DOMServer, para la cual, se ha optado por utilizar scripts de Shell ya que tiene un proceso de instalación algo más complejo como para poder utilizar ansible. Para esta máquina contamos con dos scripts, el primero de ellos se encarga simplemente de instalar las dependencias necesarias mediante APT, el segundo se encarga de realizar el proceso de instalación de DOMServer y cuenta con los pasos en el apartado de Instalación de DOMJudge. Este último script se encuentra en el Anexo, Figura 17.

Una vez ya contamos con el servidor instalado solamente queda por realizar el aprovisionamiento de los JudgeHost, para los cuales se opta al igual que en el caso del servidor, por utilizar scripts de Shell. El primero, al igual que en el servidor se encarga de instalar las dependencias necesarias mediante APT, ambas máquinas utilizan el mismo script ya que comparten dependencias, en un segundo paso, contamos con un script que se encarga de realizar la instalación del Judgehost, siguiendo los pasos descritos para ese en el apartado Instalación DOMJudge. Tras esto utilizamos un comando de aprovisionamiento de vagrant para reiniciar la máquina ya que es necesario tras realizar el último paso de instalación y, tras eso, ejecutamos un último script que se encarga de activar los servicios del sistema operativo que instala DOMJudge y que inician el JudgeHost al inicio del sistema. Los dos scripts para instalar los Judgehost se pueden encontrar en el Anexo, Figura 18 y Figura 19

Debemos asegurarnos de que los scripts y el Vagrantfile se encuentran con la siguiente estructura de carpetas:

- Vagrant/
    - provision/
        - secret/
        - inst_dep.sh
        - inst_server.sh
        - inst_*worker*_1.sh
        - inst_*worker*_2.sh
        - myslq_playbook.yml
        - Vagrantfile

Con todas las configuraciones incluidas en el Vagrantfile junto con el aprovisionamiento específico de cada máquina, solo queda iniciar las máquinas a través de Vagrant, este creará las máquinas, ejecutará el aprovisionamiento y, ya que dentro de este se ha incluido, se realizará la conexión automática entre el o los JudgeHost al servidor.

Una vez iniciado el sistema, podemos encontrar la contraseña del usuario *admin* del sistema dentro del directorio *provision/secret* en la ruta de instalación, en el fichero *initial_admin_password.secret*.

Con todo el trabajo de configuración y aprovisionamiento de las máquinas virtuales, contamos con una manera de desplegar mediante la cual con un único comando podemos iniciar todas las máquinas a la vez.

4.2 Automatización de importado de datos

Tras haber completado el despliegue del sistema contamos con otro requisito funcional, consiste en la automatización de las tareas de importado de datos (RF02). Para esta tarea se hará uso de la

API REST proporcionada por la herramienta, así como funciones propias desarrolladas en Python para convertir los datos al formato concreto esperado por la API.

Antes de comenzar con el desarrollo, es necesario comprender cómo funciona la API y cómo es el modelo de datos que reciben sus funciones. La API pública se encuentra en la dirección /api/ del servidor web de DOMJudge, aquí dentro contamos a su vez con una página en /api/doc en la cual podemos ver todas las funciones disponibles, así como cuales son los ficheros que hay que enviar en la petición, lo que responde la API y los códigos de respuesta HTTP de cada *endpoint*.

Además de la documentación exclusiva de la API, en la documentación de la herramienta se muestra el modelo de datos necesario para cada una de las funciones necesarias para importar los diferentes datos a un concurso [26]. Los datos que nos permite importar la herramienta a través de la API son los siguientes: categorías de los equipos, los equipos, las cuentas de usuario del jurado y los equipos, los datos genéricos del concurso (fechas de inicio y fin, nombre…) y los problemas del concurso.

Para la creación de algunos de los datos a través de la API se solicita utilizar un formato de fichero muy concreto llamado tsv, similar a la idea de un csv pero separado por tabulaciones, cuyo uso no es fácil debido a que es muy estricto con las tabulaciones (lo que resulta incómodo en ocasiones para localizar el origen de los fallos). Por este motivo se han desarrollado varias funciones en Python que consumen un fichero de tipo csv y que lo transforman al tsv correspondiente. Se ha optado por utilizar un csv ya que hay varios editores como por ejemplo Microsoft Excel que soportan este tipo de ficheros y de esta manera es muy sencillo de rellenar. A continuación, se describen cada una de las funciones y el formato que debe de tener el csv que consume cada una de ellas para poder realizar la transformación.

En primer lugar, contamos con las categorías o grupos (como los llama DOMJudge), esta función toma como entrada un fichero csv con una única columna en la que se indica el nombre de la categoría. Internamente se asigna un id a cada categoría y se guarda en un tsv con el formato correspondiente para DOMJudge, estos id son necesarios más adelante para asignar los equipos a cada categoría. Esta función se encuentra en el Anexo, Figura 20.

A continuación, contamos con los equipos, la función desarrollada toma como entrada un csv con cuatro columnas, el nombre del equipo, la institución a la que pertenecen, un nombre corto para la institución y la categoría a la cual pertenecen. Es importante que los nombres de las categorías coincidan exactamente con los nombres proporcionados para la primera función, ya que utiliza los nombres de las categorías para buscar su id y asignarla. La función genera un id para cada equipo, se le asigna y genera el tsv con el formato correcto para DOMJudge. La función se encuentra en el anexo, Figura 21.

Por último, se ha desarrollado la función para crear las cuentas de usuario. Esta función es algo más compleja que las anteriores ya que, además de convertir el csv a tsv, debe de generar las contraseñas para los usuarios de manera automática. Esta función toma como entrada un csv con el tipo de cuenta (judge o team), el nombre, el nombre de usuario y el equipo al que pertenecen. Al igual que las categorías en los equipos, los equipos de los usuarios deben de escribirse exactamente en ambos lugares para que la función pueda encontrar el id y asignársele. La generación de contraseñas se hace en base a una cadena con todos los caracteres disponibles y obteniendo ocho caracteres de manera aleatoria. Esta función se encuentra en el Anexo, Figura 22.

Anteriormente se han mencionado dos datos más que se pueden crear a través de la API, para estos datos no es necesaria la creación de funciones de conversión, ya que son más sencillos de

crear. Para el primer caso, los datos del concurso, simplemente se necesita un fichero yml en el cual se indican distintos parámetros, en la Figura 12 se encuentra un ejemplo de este fichero.

```
name: Final BITUCA
short-name: f-bituca
start-time: 2022-06-15T13:00:00+02:00
duration: 2:00:00
scoreboard-freeze-length: 0:30:00
penalty-time: 20
problems:
  - letter:     Prueba 1
    short-name: hello
    color:      Orange
    rgb:        '#FF7109'

  - letter:     Prueba 2
    short-name: boolfind
    color:      Forest Green
    rgb:        '#008100'
```

Figura 12. Fichero de configuración del concurso

Por último, nos encontramos con los problemas del concurso, para esto debemos contar con unos archivos comprimidos en formato ZIP en los cuales se encuentra un yml con el nombre del problema, el enunciado en pdf, y un fichero en formato Ini en el cual se indican parámetros del problema como, por ejemplo, el tiempo máximo de ejecución. La única parte que no se puede automatizar es el proceso de importado de los casos de prueba del problema, estos, deben de incluirse posteriormente a través de la interfaz, aunque no hay que ir uno por uno, sino que permite incluir todos los casos de prueba de un problema de golpe.

Una vez tenemos todos los ficheros preparados, hay que realizar las llamadas HTTP correspondientes para crear cada uno de los tipos de datos, es imprescindible seguir el siguiente orden ya que hay dependencias entre unos datos y otros:

1. Categorías
2. Equipos
3. Cuentas de usuario
4. Datos del concurso
5. Problemas del concurso

Estas llamadas las podemos realizar de varias maneras, utilizando herramientas con interfaz gráfica tales como Postman [27], que nos permite realizar llamadas HTTP, o por línea de comandos, utilizando herramientas como curl o http. En este caso se opta por realizar las llamadas utilizando la herramienta http, ya que es mediante la cual se explica el proceso en la documentación de DOMJudge.

Antes de poder realizar las llamadas, debemos generar un fichero en la carpeta del usuario llamado .netrc, en el cual debemos de configurar las credenciales del administrador de DOMJudge junto con la IP del servidor en el que se encuentra, ya que la herramienta http utiliza este fichero para obtener las credenciales de acceso a la API. Por ejemplo, para un servidor en la IP 192.168.56.104, usuario “admin” y contraseña “1234” debemos incluir la siguiente línea al fichero:

```
machine 192.168.56.104 login admin password 1234
```

Con las credenciales configuradas ya podemos realizar las llamadas utilizando http. Si deseamos realizar el importado de datos de los equipos, debemos utilizar el siguiente comando:

```
$ http --check-status -b -f POST "http://192.168.56.104/api/v4/users/groups"
tsv@to_import/groups.tsv
```

En el caso de los problemas, que no reciben un solo fichero sino varios, el comando es similar pero en el momento en el que se le indica el tipo de fichero, se indica que es una lista de ese tipo de fichero. A continuación se muestra un ejemplo:

```
$ http --check-status -b -f POST "http:// 192.168.56.104/api/v4/contests/
1/problems" "zip[]@problems/p1.zip,p2.zip"
```

En este comando se incluyen las opciones para que imprima el estado de la petición al acabar, el tipo de petición, en este caso POST, ya que es la que se utiliza para creación de objetos, el *endpoint* correspondiente y el fichero que se quiere enviar.

Para facilitar el uso de las funciones desarrolladas y de los comandos a utilizar, se ha desarrollado un script (auto.sh, disponible en el Anexo, Figura 23) que, realiza las llamadas a las funciones correspondientes para convertir los datos y, realiza las peticiones HTTP a los diferentes *endpoints* para crear los objetos. Este script solo necesita dos cosas, una de ellas es que se encuentre configurado el fichero .netrc y la otra que se siga una estructura concreta de carpetas detallada a continuación:

- Automatizacion/
  - for_import/
    - accounts.csv
    - groups.csv
    - teams.csv
  - problems/
    - problem1.zip
    - problem2.zip
  - auto.sh
  - usuarios.py

Este script imprime por pantalla el estado de las llamadas y los posibles errores que puedan generar.

5 Prueba de concepto

Para comprobar el correcto funcionamiento del sistema se realiza una prueba de concepto en la cual se prueban las automatizaciones desarrolladas, tanto el despliegue como la importación de los datos, y el correcto funcionamiento del sistema, creando un campeonato de prueba y subiendo soluciones para comprobar que funcione. Para esta prueba de concepto se ha optado por ejecutar la automatización por máquinas virtuales, junto con la automatización de importado de datos en la cual se crea un campeonato con un problema (extraído de la primera edición del campeonato) y 8 equipos pertenecientes a varias categorías e instituciones.

En primer lugar, el administrador debe de iniciar el sistema e importar los datos, por tanto, necesitamos iniciar las máquinas virtuales, para esto, con la automatización de despliegue con Vagrant y el fichero de configuración disponible en el Anexo, Figura 15, tendremos las tres máquinas desplegadas y el servidor web disponible en la dirección IP de la máquina "server", 192.168.56.102/domjudge. En la Figura 13 se muestra la interfaz web pública en la cual nos encontramos con el ranking y la posibilidad de iniciar sesión.

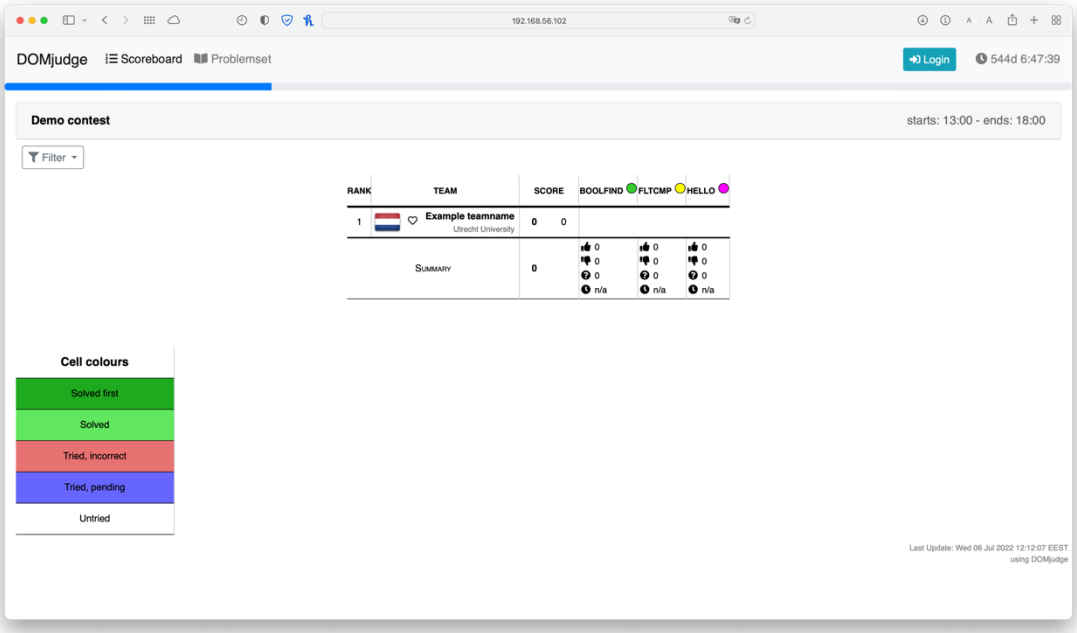

Figura 13. Interfaz web pública

Como podemos observar, DOMJudge crea un concurso de prueba con un equipo al momento de la instalación, este se puede eliminar sin mayor problema desde la consola de administración del servidor antes de crear nuestra competición.

A continuación, utilizaremos el script desarrollado para realizar la importación de datos del campeonato de forma automática. Para esto contamos con los ficheros necesarios que se describen en el apartado Automatización de importado de datos, estos son: accounts.csv, groups.csv, teams.csv, contest.yaml y los zip correspondientes a los problemas que deseamos importar al concurso. Tras ejecutar la tarea de importado de datos desde la interfaz de administrador podemos ver cómo ya tenemos disponible el campeonato, los equipos, usuarios y problemas, a través de esta misma interfaz podemos modificar datos en el caso de que sea necesario. Una vez ya

contamos con los datos del concurso, en la interfaz web ya podemos observar el nuevo ranking en el cual se muestran todos los equipos de la competición.

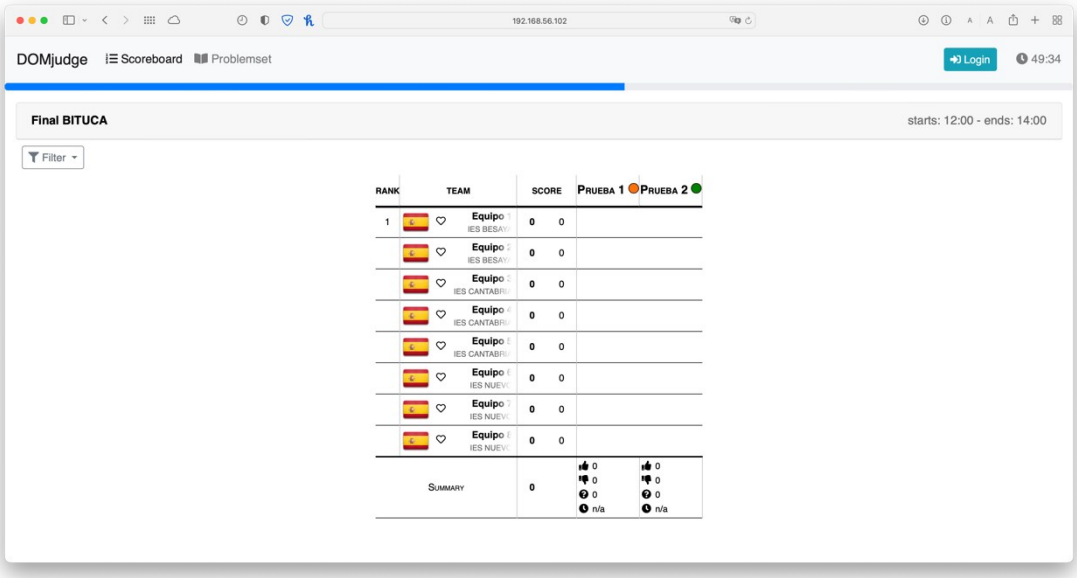

Figura 14. Interfaz web con el concurso creado

Una vez ya contamos con todos los datos del concurso, los participantes pueden iniciar sesión en el sistema con las cuentas creadas anteriormente. La interfaz web de los usuarios cuenta con un ranking en tiempo real, un apartado mediante el cual pueden hacer preguntas al administrador o al jurado, un apartado con los enunciados de los problemas propuestos y, por último, un apartado mediante el cual añaden las soluciones al sistema. Una vez los usuarios suben una solución, el sistema los ejecuta y actualiza las puntuaciones en el ranking.

En resumen, con ambas automatizaciones, simplemente tenemos que realizar unos pequeños pasos para instalar todo el sistema y conseguir un concurso desplegado. Con el Vagrantfile disponible en el Anexo, en la Figura 15, ejecutando un solo comando levantamos todas las máquinas necesarias y se realiza el proceso de instalación automáticamente en cada una de ellas. Una vez contamos con el sistema desplegado, ejecutando el script disponible en el Anexo, en la Figura 23 con los ficheros correspondientes como se explica en el capítulo 4.2, podemos crear el campeonato y ya se puede utilizar el sistema.

Esta prueba de concepto se encuentra disponible desde la red UNICAN (para acceder a la prueba desde un equipo externo hay que hacerlo a través de VPN), en la dirección http://193.144.184.128/domjudge y se puede probar el funcionamiento como participantes utilizando el usuario "*equipo1*" con contraseña "*g5a1a*2v*".

6 Bibliografía

[1] «Bituca,» [En línea]. Available: https://bituca.educantabria.es/en/.

[2] «HackerRank,» [En línea]. Available: https://www.hackerrank.com.

[3] «CMS,» [En línea]. Available: https://cms-dev.github.io.

[4] «DOMJudge,» [En línea]. Available: https://www.domjudge.org.

[5] NetApp, «¿Que son los contenedores?,» [En línea]. Available: https://www.netapp.com/es/devops-solutions/what-are-containers/.

[6] RedHat, «¿Que es una Máquina Virtual?,» [En línea]. Available: https://www.redhat.com/es/topics/virtualization/what-is-a-virtual-machine.

[7] «IOI,» [En línea]. Available: https://ioinformatics.org/journal/INFOL107.pdf.

[8] «omegaUp,» [En línea]. Available: https://ioinformatics.org/journal/v8_2014_169_178.pdf.

[9] «PC2 CSS,» [En línea]. Available: https://pc2ccs.github.io.

[10] «Mooshak,» [En línea]. Available: https://mooshak.dcc.fc.up.pt.

[11] «ICPC,» [En línea]. Available: https://icpc.global.

[12] M. P. Conlon, «RockTest: a programming contest management system,» *Journal of Computing Science in Colleges,* vol. 20, nº 5, pp. 27-35, 2005.

[13] «ECMS,» [En línea]. Available: https://www.academic-publishing.org/index.php/ejel/article/view/1465/1428.

[14] «Online Judge,» [En línea]. Available: https://onlinejudge.org.

[15] «AsyncLibrary,» [En línea]. Available: https://pypi.org/project/async-rpc/.

[16] «Tornado Web Server,» [En línea]. Available: https://www.tornadoweb.org/en/stable/.

[17] «VirtualBox,» [En línea]. Available: https://www.virtualbox.org.

[18] «APT Manual Page,» [En línea]. Available: http://manpages.ubuntu.com/manpages/trusty/es/man8/apt-get.8.html.

[19] «Kubernetes,» [En línea]. Available: https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/.

[20] «Docker,» [En línea]. Available: https://www.docker.com.

[21] «Hyper-V,» [En línea]. Available: https://docs.microsoft.com/es-es/virtualization/hyper-v-on-windows/about/.

[22] «Vagrant,» [En línea]. Available: https://www.vagrantup.com.

[23] «Docker Compose,» [En línea]. Available: https://docs.docker.com/compose/.

[24] «Ansible,» [En línea]. Available: https://www.ansible.com.

[25] «Red Hat,» [En línea]. Available: https://www.redhat.com/es.

[26] «Adding Contest Data Programmaticaly,» [En línea]. Available: https://www.domjudge.org/docs/manual/7.3/import.html.

[27] «Postman,» [En línea]. Available: https://www.postman.com.

[28] «CMS,» [En línea]. Available: https://cms-dev.github.io.

[29] «DOMJudge,» [En línea]. Available: https://www.domjudge.org.

7 Anexos

```
N=2

Vagrant.configure(2) do |config|
    config.vm.define "mysql" do |mysql|
        mysql.vm.box = "hashicorp/bionic64"
        mysql.vm.provider "hyperv" do |h|
            h.memory=2048
            h.cpus=2
        end
        mysql.vm.network :private_network, ip: "192.168.56.101"

        if Vagrant.has_plugin?("vagrant-timezone")
            mysql.timezone.value = :host
        end
        mysql.vm.provision "ansible" do |ansible|
            ansible.playbook = "provision/mysql_playbook.yml"
        end
        mysql.vm.provision "shell", reboot: true
    end
    config.vm.define "server" do |server|
        server.vm.box = "hashicorp/bionic64"
        mysql.vm.provider "hyperv" do |h|
            h.memory=2048
            h.cpus=2
        end
        server.vm.network :private_network, ip: "192.168.56.102"
        if Vagrant.has_plugin?("vagrant-timezone")
            server.timezone.value = :host
        end
        server.vm.provision "shell", path: "provision/inst_dep.sh"
        server.vm.provision "shell", path: "provision/install_server.sh", privileged:
                false
        server.vm.provision "shell", reboot: true
    end
    config.vm.define "judgehost" do |judgehost|
        judgehost.vm.box = "hashicorp/bionic64"
        mysql.vm.provider "hyperv" do |h|
            h.memory=2048
            h.cpus=2
        end
        judgehost.vm.network :private_network, type: "dhcp"
        if Vagrant.has_plugin?("vagrant-timezone")
            judgehost.timezone.value = :host
        end
        judgehost.vm.provision "shell", path: "provision/inst dep.sh"
        judgehost.vm.provision "shell", path: "provision/install_worker.sh",
               privileged: false
        judgehost.vm.provision "shell", reboot: true
        judgehost.vm.provision "shell", path: "provision/install_worker_2.sh",
               privileged: false
    end
end
```

Figura 15. Vagrantfile para despliegue en Máquinas virtuales

```
- hosts: mysql
  tasks:
    - name: Upgrade apt
      command: "apt-get update"
      ignore_errors: yes
      become: yes
    - name: Install mysql server
      become: yes
      apt:
        name:
          - mysql-server
    - name: Set MySQL root password
      command: /usr/bin/mysql -e "alter user 'root'@'localhost' IDENTIFIED
WITH mysql_native_password BY '1234';"
      ignore_errors: yes
      become: yes
    - name: Set MySQL root access
      command: /usr/bin/mysql -uroot -p1234 -e "use mysql; update user set
host='%' where user='root';"
      ignore_errors: yes
      become: yes
    - name: Set DJ database
      command: /usr/bin/mysql -uroot -p1234 -e "create database domjudge;"
      ignore_errors: yes
      become: yes
    - name: Create DJ user
      command: /usr/bin/mysql -uroot -p1234 -e "create user 'domjudge'@'%'
IDENTIFIED BY '1234';"
      ignore_errors: yes
      become: yes
    - name: Grant DJ user on db
      command: /usr/bin/mysql -uroot -p1234 -e "grant all privileges on
domjudge.* to 'domjudge'@'%';"
      ignore_errors: yes
      become: yes
    - name: Enable remote access
      command: sed -i 's/bind-address   = 127.0.0.1/bind-address    =
192.168.56.101/g' "/etc/mysql/mysql.conf.d/mysqld.cnf"
      ignore_errors: yes
      become: yes
```

Figura 16. Ansible Playbook para aprovisionamiento de MySQL

```
#!/bin/bash

cd /home/vagrant

su vagrant

wget https://www.domjudge.org/releases/domjudge-7.3.4.tar.gz

tar -xzf domjudge-7.3.4.tar.gz

cd domjudge-7.3.4

./configure --prefix=$HOME/domjudge
make domserver
sudo make install-domserver

/home/vagrant/domjudge/domserver/bin/dj_setup_database genpass
cat /home/vagrant/domjudge/domserver/etc/dbpasswords.secret
echo "dummy:192.168.56.101:domjudge:domjudge:1234" >
/home/vagrant/domjudge/domserver/etc/dbpasswords.secret
/home/vagrant/domjudge/domserver/bin/dj_setup_database -u root -p 1234
install

sudo cp /home/vagrant/domjudge/domserver/etc/apache.conf /etc/apache2/conf-
available/domjudge.conf
sudo cp /home/vagrant/domjudge/domserver/etc/domjudge-fpm.conf
/etc/php/7.2/fpm/pool.d/domjudge.conf
sudo a2enmod proxy_fcgi setenvif rewrite
sudo a2enconf php7.2-fpm domjudge
sudo service php7.2-fpm reload
sudo service apache2 reload

cp /home/vagrant/domjudge/domserver/etc/restapi.secret
/vagrant/provision/secret/.
cp /home/vagrant/domjudge/domserver/etc/initial_admin_password.secret
/vagrant/provision/secret/.
```

Figura 17. Script instalación DOMServer

```
#!/bin/bash

cd /home/vagrant

su vagrant

wget https://www.domjudge.org/releases/domjudge-7.3.4.tar.gz

tar -xzf domjudge-7.3.4.tar.gz

cd domjudge-7.3.4

./configure --prefix=$HOME/domjudge
make judgehost
sudo cp /vagrant/provision/secret/restapi.secret /home/vagrant/domjudge-7.3.4/etc
sudo sed -i 's/localhost/192.168.56.102/g' "/home/vagrant/domjudge-
7.3.4/etc/restapi.secret"
sudo make install-judgehost

sudo useradd -d /nonexistent -U -M -s /bin/false domjudge-run

sudo cp /home/vagrant/domjudge/judgehost/etc/sudoers-domjudge /etc/sudoers.d/.

cd /home/vagrant/domjudge/judgehost/bin
sudo ./dj_make_chroot -y

sudo sed -i 's/GRUB CMDLINE LINUX DEFAULT="quiet"/GRUB CMDLINE LINUX DEFAULT="quiet
cgroup_enable=memory swapaccount=1"/g' "/etc/default/grub"

sudo update-grub
```

Figura 18. Primer Script para instalación de JudgeHost

```
#!/bin/bash
sudo sed -i 's,ExecStart=/home/vagrant/domjudge/judgehost/bin/judgedaemon -n
0,ExecStart=/home/vagrant/domjudge/judgehost/bin/judgedaemon,g'
"/home/vagrant/domjudge/lib/systemd/system/domjudge-judgehost.service"
sudo ln domjudge/lib/systemd/system/* /etc/systemd/system/

sudo systemctl enable create-cgroups --now

sudo systemctl enable domjudge-judgehost
sudo systemctl start domjudge-judgehost
```

Figura 19. Segundo Script para instalación de JudgeHost

```
def groups(input, output):
    path = os.getcwd()
    csv_groups_import = open(os.path.join(path, str(input)))
    groups = open(os.path.join(path, str(output)), "w")
    reader = csv.reader(csv_groups_import)
    writer = open(os.path.join(path, "tmp/groups.csv"), "w")
    id=random.randint(200, 10000)
    groups.write('File_Version\t1\n')
    writer.write('ID, NAME\n')
    next(reader)
    for row in reader:
        name=row[0]
        groups.write(str(id)+'\t'+name+'\n')
        writer.write(str(id)+','+name+'\n')
        id=id+10
```

Figura 20. Función de conversión Categorías

```
def teams(input, output):
    path = os.getcwd()

    csv_teams_import = open(os.path.join(path, str(input)))
    csv_teams_export = open(os.path.join(path, str(output)),"w")
    groups_imported = open(os.path.join(path, "tmp/groups.csv"))

    writer = open(os.path.join(path, "tmp/teams.csv"), "w")
    writer.write('ID, NAME\n')

    groups = csv.reader(groups_imported)
    teams = csv.reader(csv_teams_import)

    id=random.randint(200, 10000)

    next(teams)
    csv_teams_export.write("File_Version\t2\n")

    groups2=pd.read_csv("to_import/groups.tsv")

    for row in teams:
        groups_imported.seek(0)
        next(groups)
        id=id+10
        ext_id=''
        institution=row[1]
        cat=row[3]
        for row2 in groups:
            if row2[1] == cat:
                group=row2[0]
        name=row[0]
        inst_short=row[2]
        country='ESP'
        ext_inst=''
        csv_teams_export.write(str(id) + "\t" + ext_id + "\t" + group + "\t" + name +
"\t" + institution + "\t" + inst_short + "\t" + country + "\t" + ext_inst + "\n")
        writer.write(str(id)+','+name+'\n')
```

Figura 21. Función de conversión Equipos

```
def accounts(input, output):
    str_pass = str("qwertyuiopasdfghjklñzxcvbnmQWERTYUIOPASDFGHJKLÑZXCVBNM*?¿¡!$%&")
    path = os.getcwd()
    csv_users_import = open(os.path.join(path, str(input)))
    groups = open(os.path.join(path, str(output)), "w")
    reader = csv.reader(csv_users_import)
    id=random.randint(200, 10000)
    groups.write('accounts\t1\n')
    next(reader)

    teams_imported = open(os.path.join(path, "tmp/teams.csv"))
    teams = csv.reader(teams_imported)

    for row in reader:
        print('\nNEW')
        tipo=row[0]
        name=row[1]
        password = ""
        team = row[3]
        print('Team: ',team)
        if team=="null":
            team_id=''
        else:
            print('Equipos...')
            teams_imported.seek(0)
            for row2 in teams:
                print(row2[1])
                if row2[1] == team:
                    print(row2[1],team)
                    team_id=row2[0]
        if tipo=='team':
            username='team-'+team_id
        else:
            username=row[2]
        for i in range(8):
            password = password + random.choice(str_pass)
        groups.write(tipo+'\t'+name+'\t'+username+'\t'+password+'\n')
```

Figura 22. Función de conversión Cuentas de usuarios

```
#!/bin/bash

# IMPORTANTE
# DECLARAR VARIABLE CON LA DIRECCION DEL SERVIDOR
SERVER=localhost:3000

#Check temporary files for running the script
if ! [ -d "to_import" ]; then
    mkdir to_import
fi
if ! [ -d "tmp" ]; then
    mkdir tmp
fi
cd to_import
if ! [ -f "groups.tsv" ]; then
    touch groups.tsv
fi
if ! [ -f "teams.tsv" ]; then
    touch teams.tsv
fi
if ! [ -f "users.tsv" ]; then
    touch users.tsv
fi
cd ..
cd tmp
if ! [ -f "groups.csv" ]; then
    touch groups.csv
fi
cd ..
echo "\nDOMJUDGE DATA IMPORT\n"
#Import groups
echo "Setting up Team categories..."
python3 -c 'import usuarios as u; u.groups("for_import/groups.csv",
"to_import/groups.tsv")'
echo "Importing Team categories...\n"
http --check-status -b -f POST "http://$SERVER/api/v4/users/groups"
tsv@to_import/groups.tsv

#Import Teams
echo "Setting up Teams..."
python3 -c 'import usuarios as u; u.teams("for_import/teams.csv",
"to_import/teams.tsv")'
echo "Importing Teams..."
http --check-status -b -f POST "http://$SERVER/api/v4/users/teams"
tsv@to_import/teams.tsv

#Import Users
echo "Setting up Teams..."
python3 -c 'import usuarios as u; u.accounts("for_import/accounts.csv",
"to_import/accounts.tsv")'
echo "Importing Teams..."
http --check-status -b -f POST "http://$SERVER/api/v4/users/accounts"
tsv@to_import/accounts.tsv

#Import Contest data
echo "Importing Teams..."
cont_id=$(http --check-status -b -f POST "http://$SERVER/api/v4/contests"
yaml@to_import/contest.yaml)
echo "Contest id: $cont_id"

#Import problems
echo "Importing problems"
http --check-status -b -f POST "http://$SERVER/api/v4/contests/$cont_id/problems"
"zip[]@problems/hello2.zip"

rm -R tmp
```

Figura 23. Script de automatización de importado de datos