



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE MÁSTER

Sistema de gestión integral de una flota de vehículos operativos

Máster Universitario en Dirección TIC para la Defensa
Especialidad de Sistemas y Tecnologías de la Información

ALUMNO: Rafael Martínez Mesones

DIRECTORES: Norberto Fernández García

CURSO ACADÉMICO: 2022-2023

Universida_{de}Vigo

RESUMEN

Las Unidades Operativas de Investigación sustentan gran parte de su trabajo en la flota de vehículos que tienen asignados para la realización de las labores que les son encomendadas.

Los suministros de estos medios de transporte suelen tener su origen en compras realizadas por el Organismo competente, en las autorizaciones judiciales de los vehículos incautados a los objetivos de una operación, en la cesión temporal de otras Unidades u Organismos, etc.

Todos estos medios se categorizan y se clasifican en relación a sus características: tipo de vehículo, cilindrada, kilómetros recorridos, antigüedad, color, etc., estableciéndose varias categorías donde ubicarlos.

Se necesita registrar y documentar todos los datos que generan estos vehículos durante su vida útil, como pueden ser: kilómetros recorridos mensualmente, reparaciones, revisiones mecánicas, accidentes, sustituciones de piezas, inspecciones ITV, seguros, etc., información que posteriormente puede ser explotada con procesos “de minería de datos” con el objetivo de predecir fechas de mantenimientos, recambios e inspecciones.

El principal problema se plantea a la hora de realizar una adjudicación de los elementos de transporte a los distintos Departamentos que componen una Unidad Operativa de una forma equilibrada y equitativa, sin que nadie se sienta perjudicado o desfavorecido en comparación al resto de Grupos o Departamentos.

El principal objetivo del presente trabajo es el diseño de una herramienta informática con la que administrar y gestionar este parque móvil, sus reparaciones, repostajes, accidentes, etc., que será fundamental para llevar un perfecto control de todos los elementos de transporte existentes.

Dada la importancia de esta herramienta, el sistema se sustenta en una arquitectura que puede escalar horizontalmente, en relación al número de usuarios y peticiones de proceso, realizando un despliegue en contenedores “Docker” y utilizando el orquestador “Kubernetes” para gestionar toda la plataforma.

PALABRAS CLAVE

Vehículos, operativo, Docker, Kubernetes, Bootstrap, PHP, gestión integral

AGRADECIMIENTOS

A mi mujer y mis hijos, que tanto me han apoyado a lo largo de toda mi vida.

A mis compañeros de máster, con los que he compartido momentos inolvidables.

A mi tutor, por su infinita paciencia y dirección de este trabajo.

CONTENIDO

Contenido	1
Índice de Figuras	3
Índice de Tablas.....	5
1 Introducción y objetivos	7
1.1 Descripción y objetivos.....	7
1.2 Motivación del proyecto	8
1.3 Antecedentes y estado actual	9
1.4 Organización de la memoria	9
2 Estado del arte	11
2.1 Introducción	11
2.1.1 Concepto de contenedor	11
2.1.2 Características.....	11
2.2 Nacimiento y evolución de los contenedores.....	12
2.2.1 La primera jaula. - chroot	12
2.2.2 Contenedores LXC	12
2.2.3 Aparición de Docker.....	13
2.3 Diferencias entre los distintos tipos de contenedores	15
2.4 Orquestación de contenedores	16
2.4.1 Introducción	16
2.4.2 Docker Bundle	16
2.4.3 Kubernetes	17
2.5 Herramientas para la orquestación de <i>Kubernetes</i>	19
2.5.1 Red Hat OpenShift.....	19
2.5.2 Kubectl.....	21
2.5.3 Otros administradores de cluster para Kubernetes	21
2.6 Software para la administración y automatización de procesos. Ansible	22
2.7 Desarrollo de la aplicación web	22
2.7.1 BootStrap	23
2.7.2 PHP	24
2.7.3 Nginx	24
2.7.4 MariaDB	25
2.8 Selección y justificación del entorno hardware / virtual / software	25
3 Desarrollo del TFM	27

3.1 Diseño de la infraestructura	27
3.1.1 Hardware y Software Base	27
3.1.2 Configuración virtual.....	28
3.1.3 Arquitectura y conectividad.....	29
3.2 Implantación e instalación del clúster Kubernetes.....	29
3.2.1 Despliegue del cluster Kubernetes mediante Ansible.....	29
3.2.1 Comprobando el estado del cluster	36
3.2.2 Configuración repositorio NFS común al cluster	37
3.2.3 Creando volúmenes persistentes.....	39
3.3 Diseño de la aplicación	42
3.3.1 Requisitos y ámbito de aplicación	42
3.3.2 Establecimiento de requisitos	43
3.3.1 Actores y especificaciones de los casos de uso	45
3.3.2 Descripción de los casos de uso.....	46
3.3.3 Procedimiento Adjudicación de vehículos.	55
4 Resultados / Validación.....	58
4.1 Despliegue de la plataforma.....	58
4.1.1 Despliegue servicio Web	58
4.1.2 Despliegue sistema gestor de base de datos	61
4.2 Modelo lógico de datos	65
4.3 Despliegue de la aplicación.....	66
4.3.1 Ficheros PHP de la aplicación y de configuración	66
4.3.2 Procedimiento de clasificación de los vehículos	71
4.3.3 Procedimiento de adjudicación de los vehículos	75
5 Conclusiones y líneas futuras	83
5.1 Conclusiones	83
5.2 Líneas futuras	84
6 Bibliografía.....	87
Anexo I: Código fuente del procedimiento de clasificación de vehículos	90
Anexo II: Código fuente del procedimiento de adjudicación de vehículos.....	95

ÍNDICE DE FIGURAS

Figura 2-1 Tabla de Procesos kernel Linux.....	14
Figura 2-2 Procesos aislados en namespaces independientes. Autoría propia.....	15
Figura 2-3 Orquestación en Docker.	16
Figura 2-4. Tipos de despliegues de aplicaciones	18
Figura 2-5. Arquitectura de un cluster Kubernetes. Autoría propia.	19
Figura 2-6. Descripción general de la arquitectura Red Hat OpenShift 4. (tomada de [27]).....	20
Figura 2-7. Modelo MVC.....	23
Figura 2-8. Funcionamiento NGINX con PHP-FPM.	25
Figura 3-1. Servidor HPE Proliant DL160 – Gen9.	27
Figura 3-2. Esquema conectividad e infraestructura. Autoría propia.....	29
Figura 3-3. Generando par de claves RSA para conexiones SSH. Autoría propia.	30
Figura 3-4. Instalación clave pública del servidor “zelenski”. Autoría propia.	30
Figura 3-5. Información del clúster definido en el proyecto. Autoría propia.	36
Figura 3-6. Información sobre la ejecución de PODS en el <i>cluster</i> . Autoría propia.....	37
Figura 3-7. Información total de los “namespaces” y PODS en el <i>cluster</i> . Autoría propia.	37
Figura 3-8. Información de los servicios actualmente desplegados en el <i>cluster</i> . Autoría propia.	37
Figura 3-9. Conectividad VMs con repositorio NFS. Autoría propia.	38
Figura 3-10. Contenido fichero <i>fstab</i> del servidor <i>putin</i> . Autoría propia.	39
Figura 3-11. Estado creación volumen <i>nfswww</i> en el cluster Kubernetes. Autoría propia.	41
Figura 3-12. Volúmenes persistentes creados en el cluster Kubernetes. Autoría propia.	42
Figura 3-13. Entidades cuya información procesa el sistema. Autoría propia.....	43
Figura 3-14. Casos de Uso global del sistema. Autoría propia.	46
Figura 3-15. Procedimiento adjudicación nuevo vehículo. Autoría propia.....	57
Figura 4-1 Despliegue contenedores nginx. Autoría propia.....	60
Figura 4-2 Despliegue contenedores nginx. Autoría propia.....	61
Figura 4-3 Despliegue contenedores nginx y MariaDB. Autoría propia.....	63
Figura 4-4 Servicios balanceados nginx y MariaDB. Autoría propia.	64
Figura 4-5 Modelo base de datos vehículos. Autoría propia.....	65
Figura 4-6 Modelo base de datos vehículos. Autoría propia.....	66
Figura 4-7 Pantalla Login de acceso a la aplicación. Autoría propia.	67
Figura 4-8 Código PHP registro sesión usuario. Autoría propia.....	67
Figura 4-9 Validación credenciales usuario. Autoría propia.....	68
Figura 4-10 Función registro trazas de auditoría. Autoría propia.	68

Figura 4-11 Página principal de la aplicación. Autoría propia.....	69
Figura 4-12 Secciones que componen las vistas de la aplicación. Autoría propia.....	70
Figura 4-13 Mensajes del Centro de Alertas. Autoría propia.....	70
Figura 4-14 Pantalla de gestión de vehículos. Autoría propia.	71
Figura 4-15 Nuevos vehículos introducidos en el sistema. Autoría propia.....	73
Figura 4-16 Clasificación de los nuevos vehículos. Autoría propia.....	74
Figura 4-17 Organigrama de organización de la Unidad. Autoría propia.....	76
Figura 4-18 Resultado final de la adjudicación de los vehículos. Autoría propia.....	79

ÍNDICE DE TABLAS

Tabla 2-1 Características Red Hat OpenShift (tomada de [27]).....	20
Tabla 3-1 Recursos Hardware Virtuales y S.O. VM Zelenski	28
Tabla 3-2 Recursos Hardware Virtuales y S.O. VMs <i>cluster</i> Kubernetes	28
Tabla 3-3 Recursos definidos en los volúmenes “nfs”. Autoría propia.	38
Tabla 3-4 Recursos definidos en el volumen “nfs – Servidor Web”. Autoría propia.	40
Tabla 3-5 Recursos definidos en el volumen “nfswww-pvc”. Autoría propia.....	41
Tabla 3-6 Recursos definidos en el volumen “nfs- servidor Base de Datos”. Autoría propia.	41
Tabla 3-7 Recursos definidos en el volumen “nfsbddd-pvc”. Autoría propia.	42
Tabla 3-8 Recursos funcionales de la aplicación. Autoría propia.	44
Tabla 3-9 Recursos no funcionales de la aplicación. Autoría propia.	44
Tabla 3-10 Definición casos de uso del usuario avanzado. Autoría propia.	52
Tabla 3-11 Definición casos de uso del usuario administrador. Autoría propia.	55
Tabla 3-12 Puntuación puestos de trabajo. Autoría propia.	55
Tabla 3-13 Ejemplo de Unidad, plantilla y catálogo de los puestos de trabajo. Autoría propia.	56
Tabla 3-14 Factores para la clasificación de los vehículos. Autoría propia.	56
Tabla 4-1 Baremos de clasificación de los vehículos. Autoría propia.	72
Tabla 4-2 Tabla de clasificación de categorías.....	73
Tabla 4-3 Tabla comprobación procedimiento de clasificación. Autoría propia.	75
Tabla 4-4 Tabla Catálogo puestos de trabajo de la Unidad. Autoría propia.	76
Tabla 4-5 Tabla Definición puestos de trabajo. Autoría propia.	77
Tabla 4-6 Puntuación puestos del catálogo. Autoría propia.....	77
Tabla 4-7 Simulación cálculo porcentaje reparto vehículos categoría “A”. Autoría propia.....	78
Tabla 4-8 Cálculo porcentaje final para adjudicación vehículos categoría “A”. Autoría propia.	78
Tabla 4-9 Adjudicación segundo vehículo categoría “A”. Autoría propia.	80
Tabla 4-10 Adjudicación tercer vehículo categoría “A”.	80
Tabla 4-11 Tabla porcentajes de reparto para la novena adjudicación. Autoría propia.	81

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Descripción y objetivos

La gestión de una flota de vehículos comprende la administración y organización de todos los elementos de transporte asignados a una Unidad/Organismo. Con la ayuda de la informática, podemos controlar los vehículos de una flota con la finalidad de realizar un control holístico de la misma, al mismo tiempo que pueda aumentarse la seguridad de los conductores y se reduzcan los riesgos de accidentes o siniestros con motivo de su utilización.

Con este sistema se pretenden aumentar la eficiencia y la productividad, al tiempo que se mejora la seguridad de los conductores y se optimiza el estado del funcionamiento de los vehículos. Esto puede lograrse combinando el seguimiento diario con informes sobre el consumo de combustible, kilometraje acumulado, supervisión de la actividad de los conductores y la gestión y mantenimiento de todos los elementos mecánicos de estos medios de transporte.

El primer objetivo de este trabajo es diseñar un sistema integral, basado en tecnologías de virtualización y contenerización, en el que se puedan ejecutar las aplicaciones informáticas de gestión de la flota de vehículos en contenedores que puedan escalar en función de la concurrencia de usuarios, de la ejecución de procesos de automatización, etc. Analizaremos las distintas herramientas de contenerización disponibles y nos centraremos en la instalación y configuración de una infraestructura basada en contenedores Docker [1] y en su orquestación mediante Kubernetes [2].

Una vez configurada la infraestructura física, realizaremos un diseño de los distintos componentes software que formarán parte de la aplicación, como pueden ser, la arquitectura de tablas y relaciones de la base de datos, los requisitos funcionales que debe contemplar la aplicación, el desarrollo de un procedimiento con el que poder adjudicar, de forma automática, los vehículos pendientes de asignación, la ejecución de tareas automáticas para simplificar el trabajo de los usuarios y finalmente, la interacción con la que deben comunicarse los distintos módulos entre sí para tener actualizada la información de la flota de vehículos. Uno de los desafíos de esta gestión lo constituye el procedimiento de adjudicación, cuyo funcionamiento debe ser transparente para todos los Departamentos de la Organización, imparcial a la hora de establecer los cupos y que pueda ser auditado por los usuarios, de forma que no sea parcial, manipulable o improcedente el resultado de su proceso.

Puesto que estamos transitando los comienzos de la quinta era, en la que elementos como la robótica, inteligencia artificial, nanotecnología y la conectividad total son elementos que se están convirtiendo en

cotidianos, podría completarse el sistema con el desarrollo de una aplicación para dispositivos móviles, con la que los usuarios, durante sus desplazamientos, puedan interactuar con la gestión de vehículos y puedan enviar información, en tiempo real, de algunos aspectos que son de interés para la Organización.

El segundo objetivo del proyecto es diseñar, así como desarrollar el embrión de una aplicación web, que se ejecute en el entorno virtualizado, programada en el lenguaje PHP [3], que cuenta con un *back-end* de base de datos, basado en MariaDB [4] y, cuya finalidad, sea la gestión integral de una flota de vehículos en una Unidad Operativa.

Teniendo en cuenta esta finalidad, deberemos abordar los siguientes objetivos específicos:

- Diseñar un entorno virtualizado, con suficiente capacidad de cómputo y almacenamiento, capaz de dar servicio a todos los usuarios activos de la Unidad.
- Analizar las distribuciones de Kubernetes y contenedores de Docker que cumplan con los requerimientos y ofrezcan una solución a las necesidades del proyecto.
- Plantear una arquitectura de producción, dirigida por un orquestador, con el que llevar a cabo las tareas de despliegue y monitorización de contenedores y aplicaciones.
- Estudiar los requerimientos, especificaciones y circunstancias comprendidas en la gestión de la flota de vehículos.
- Diseñar una aplicación web que proporcione una mayor agilidad en la actividad diaria del departamento encargado de la gestión de una flota de vehículos.

1.2 Motivación del proyecto

Las Unidades operativas soportan, en su día a día, una alta carga de trabajo en la gestión de su material y, sobre todo, en la administración, gobierno y registro de trámites que deben efectuar con la flota de vehículos que tienen asignados.

Si bien hoy en día la utilización de dispositivos informáticos, con los que procesar y almacenar la información, se ha convertido en una herramienta básica y con la que trabajan la mayor parte de las Unidades, diversos procedimientos y actividades, que pueden ser calificadas como “rutinarias”, se realizan repetidamente y penalizan el rendimiento y producción del Departamento encargado de llevar a cabo estas gestiones.

Actualmente, las Unidades Operativas no cuentan con una herramienta informática en la que apoyarse para llevar a cabo las funciones encomendadas, por lo que cada Unidad, de forma individual, ha desarrollado métodos o sistemas de trabajo individuales y aislados que no son suficientes para gestionar la flota de vehículos.

Se pretende obtener un beneficio en la productividad de la Unidad, reduciendo procedimientos pautados y repetitivos y ofreciendo un entorno centralizado donde poder gobernar la flota de vehículos, con información en tiempo real y actualizada.

La mayoría de los procesos utilizados en la gestión de una flota de medios de transporte pueden programarse mediante “procedimientos almacenados” y “micro-servicios”, lo que repercute en un ahorro de tiempo y en la automatización de procedimientos rutinarios. Un ejemplo de estos procedimientos rutinarios es realizar la catalogación de los vehículos en función de diversos parámetros, como son, kilometraje, KW, color, tipo de vehículo, etc. En este proyecto se desarrollará un procedimiento automático para realizar la clasificación de los nuevos vehículos asignados.

Uno de los hitos fundamentales del sistema lo compone el desarrollo de un módulo inteligente de “adjudicación” de los medios de transporte a los distintos Departamentos y Grupos, que componen la

Unidad, de forma que el reparto sea equitativo, neutral e íntegro de acuerdo a unos parámetros configurables y definidos de antemano. Estos elementos de calificación, fijados por la Dirección, están basados en la composición del número de agentes de cada Departamento, misión que desempeñan, especialización, funciones encomendadas, etc. En el proyecto se lleva a cabo la programación de un procedimiento para la adjudicación automática de los nuevos vehículos que se incorporen a la Unidad, de forma que sea proporcional a las catalogaciones definidas por la Dirección.

En la actualidad, las empresas y organizaciones están trasladando sus sistemas de producción a entornos virtualizados y contenerizados, dotados de una mayor portabilidad y con los que poder escalar y dar servicio ininterrumpido a los usuarios de una forma mucho más eficiente. En este proyecto se contempla, en la implantación de la arquitectura, el uso de contenedores, los cuales pueden ejecutarse en entornos heterogéneos y adaptarse a las capacidades hardware de la Unidad.

1.3 Antecedentes y estado actual

Actualmente, las Unidades Operativas no cuentan con una herramienta integral que proporcione todos estos servicios; algunas utilizan documentos Excel con los que llevar un control de estos medios de transporte; otras utilizan aplicaciones desarrolladas por personal aficionado a la informática, en ordenadores personales, con los que automatizan algunos de los servicios descritos; otras se basan en aplicaciones desarrolladas en entornos de la aplicación Microsoft Access, que implementan varios de los requisitos descritos anteriormente, pero no llegan a completar la totalidad de las necesidades expuestas.

Desde los inicios de la informática hemos observado un gran número de procedimientos para desplegar aplicaciones en los distintos sistemas, las cuales proporcionaban una serie de ventajas e inconvenientes respecto de las demás. En los primeros tiempos se realizaban instalaciones en servidores que debían ser configurados manualmente, lo que repercutía en operaciones más lentas y donde un fallo (habitualmente humano), impedía revertir el proceso y era necesario comenzar desde el principio inexorablemente. Además de esta desventaja, las máquinas donde se realizaban estas tareas debían ser muy potentes y no se aprovechaba al máximo su capacidad y disponibilidad. Posteriormente se implementaron tecnologías basadas en “contenedores”, lo que permitió remediar parte del problema antes descrito y simplificar este cometido, si bien los servidores no trabajaban al máximo de rendimiento y se percibía una baja disponibilidad en su utilización. Para mejorar estos problemas surgió la tecnología de contenedores Docker, la cual permite el despliegue de los mismos de una forma más sencilla de lo que venían siendo habitual, resolviéndose la mayoría de problemas descritos anteriormente. Y, finalmente, una vez que este despliegue creció de forma exponencial y los contenedores se incrementaron en un número elevado dentro de un sistema informático, se hizo necesario contar con orquestadores que permitieran llevar a cabo labores de administración y gestión de forma centralizada, como es el caso de Kubernetes. Además de todo lo mencionado, Kubernetes lleva a cabo una labor de equilibrio de carga, que permite resolver el problema de la baja disponibilidad, y controla el estado de ejecución de los servidores, lo que le permite reconfigurar las peticiones ante el fallo de una máquina.

1.4 Organización de la memoria

En el Capítulo 2, comenzaremos este trabajo realizando un recorrido por las distintas tecnologías que se desean utilizar en su implementación, evaluando cada una de ellas en el contexto y época actual, además de realizar un breve recorrido en su evolución desde el nacimiento de cada una de ellas.

Como se ha comentado anteriormente, una de las tecnologías que se emplean en el proyecto es la de contenedores. Se han estudiado los distintos tipos que existen en el mercado y se han evaluado los pros y contras de cada uno de ellos, llegando a la elección del más indicado para este proyecto.

Las tecnologías de contenerización avanzan y se renuevan continuamente, por lo que se ha incorporado un sistema de orquestación para la administración, control y despliegue de los elementos de la aplicación. En el Capítulo 2, apartados 4 y 5, se ha realizado una prospección de los sistemas existentes en el mercado y se han evaluado los más utilizados y populares en la comunidad informática.

En cuanto a la metodología de diseño y programación de la aplicación, se ha utilizado el modelo *MVC* (*Modelo-Vista-Controlador*), utilizando un lenguaje de programación para la parte “controlador”, varios elementos *CSS* y *Javascript* para abordar la apariencia y composición de las páginas web y, por último, una base de datos *open source* donde almacenar la información.

En el Capítulo 3, una vez abordados todos estos aspectos, pasaremos a la implementación de todos los elementos descritos en un sistema hardware, configurando un entorno de ejecución que contengan elementos de cómputo y de almacenamiento en red, como puede ser un repositorio NFS (*Network File System*) donde almacenar los datos comunes de la aplicación, así como una red de comunicación donde puedan conectarse y comunicarse todos los sistemas.

Cuando este entorno hardware y de comunicación se encuentre perfectamente configurado, se instalarán los sistemas operativos, máquinas virtuales y sistemas de contenerización con los que llevar a cabo el despliegue de la aplicación. Para agilizar los despliegues y configuraciones de los servidores virtuales y de los sistemas de contenedores, se utiliza una herramienta con la que preparar la infraestructura subyacente de hosts y servidores mediante scripts de ejecución automática que pueden ser ejecutados en una granja de sistemas de forma automática y descentralizada.

El siguiente objetivo es el despliegue y configuración de los contenedores con los elementos que necesita la aplicación, comenzando por el servidor web, el preprocesador PHP y de la base de datos. Una vez configurados y desplegados, se instalarán los elementos *CSS* (*Cascading Style Sheets*), con los que componer y estructurar las páginas web, así como librerías JavaScript con las que programar varias funciones interactivas utilizadas en la aplicación.

Con la infraestructura hardware y software perfectamente configurada y en ejecución, comenzaremos a definir los requisitos, actores y especificaciones de uso de la aplicación, describiendo y analizando los procedimientos más complejos que hay que desarrollar.

En el Capítulo 4 pasaremos a desarrollar, implementar y comprobar el funcionamiento de dos procedimientos que se han desplegado y programado en el proyecto: clasificación automática de vehículos, así como el reparto y adjudicación de los mismos a los Departamentos y Grupos de la Unidad.

Finalmente, en el Capítulo 5, finalizamos este proyecto con una serie de conclusiones que se han observado a lo largo de su elaboración y con la exposición de distintas líneas para la continuación del mismo.

2 ESTADO DEL ARTE

Vamos a realizar un breve recorrido por las distintas tecnologías y soluciones relacionadas con este trabajo, comenzando con la evolución que han sufrido los contenedores desde la primera llamada al entorno *chroot()*, pasando por las llamadas “jaulas”, continuando por los contenedores LXC y finalizando en los hoy conocidos contenedores Docker. Finalmente abordaremos los distintos orquestadores con los que podemos trabajar en una infraestructura basada en contenedores.

2.1 Introducción

2.1.1 Concepto de contenedor

Un contenedor es la “unidad básica” que nos permite ejecutar una aplicación. Una unidad que dispone del software mínimo necesario como son las librerías, herramientas del Sistema Operativo y los programas ejecutables de la aplicación, así como tiempo de ejecución.

2.1.2 Características

Las características más importantes son [5]:

- Liviano, ocupa poco espacio y sólo incorpora el software mínimo necesario para poder ejecutar la aplicación. Comparte el *kernel* del sistema operativo anfitrión, por lo que no necesita un sistema operativo independiente en cada contenedor.
- Instanciable, es decir, con una sola imagen pueden instanciarse varios contenedores al mismo tiempo en ejecución.
- Aislado, se ejecuta en su espacio de usuario y cuentan con su propio identificador de proceso en el sistema operativo base.
- Estándar, puede portarse y ejecutarse tanto en servidores Linux (cualquier distribución) como en servidores Windows. Esta característica posibilita el migrar o ejecutar contenedores en distintos sistemas hardware, distintos sistemas operativos, virtualizadores, etc.
- Control de versiones y restauración, posibilitando versionar los distintos cambios que se han realizado en su desarrollo. Puede regresarse a una versión anterior en cualquier momento.

- Rápida implementación, proporcionando el rápido despliegue de una aplicación en un tiempo mínimo (segundos). Esta característica constituye una gran ventaja respecto de los despliegues tradicionales, que podrían tardar días.
- Mayor densidad de proceso, aprovechando al máximo las características del hardware base al ejecutar múltiples contenedores al mismo tiempo, cada uno de los cuales se adapta a una carga de trabajo específica y proporciona una mayor eficiencia.

2.2 Nacimiento y evolución de los contenedores

2.2.1 La primera jaula. - *chroot*

Según comenta Gunnar Wolf en el artículo: “Contenedores: Un poco de historia” [6] , en el año 1982, Bill Joy, uno de los arquitectos del sistema Unix de BSD y fundador de Sun Microsystems, realizó una llamada al sistema *chroot*, con la que consiguió un entorno aislado donde poder llevar a cabo sus pruebas de desarrollo sin interferir en el resto de procesos y recursos de su sistema operativo principal. Con este procedimiento, el sistema operativo ofrecía a un proceso una vista parcial del sistema, limitándolo a un directorio indicado.

Este sistema está ideado para proveer cierto aislamiento, tanto en cuanto existen diversos métodos que pueden emplearse para “escapar” de este enjaulamiento, además de que el funcionamiento del mismo se condiciona a restringir la vista del sistema de archivos, sin proteger otros aspectos como pueden ser la lista de procesos, la configuración de las tarjetas de red, o el acceso a los dispositivos del sistema.

El primer sistema operativo que soportó esta funcionalidad, conocida como “jaulas” (*jails*), fue la versión 4 de FreeBSD, que fue liberada en el año 2.000.

Posteriormente, otros sistemas operativos, como son Solaris, Linux y Windows, incorporaron esta funcionalidad; tanto en Linux como en Windows lo hicieron mediante productos de terceros, como fueron: Parallels Virtuozzo Containers para Windows, Vserver y OpenVZ para Linux.

Vserver [7] vio la luz en el año 2.001, a través del proyecto liderado por Jacques Gélinas, introduciendo en el S.O. Linux la implementación de un entorno aislado. Esta base, que permitió la creación de múltiples espacios de usuario en Linux, puede considerarse como el origen de los actuales contenedores.

2.2.2 Contenedores LXC

Los contenedores LXC [8] surgieron de un proyecto público, creado en Github [9] en el año 2.013, a partir del cual fue evolucionando hasta su versión actual, la 5.0.1, publicada en junio del presente año.

Como se describe en su página principal [9], este tipo de contenedores se consideran elementos intermedios entre una jaula *chroot* y una máquina virtual completa. El objetivo es la creación de un entorno lo más parecido posible a una instalación estándar de Linux, pero sin la necesidad de contar con un *kernel* separado.

Comparten el mismo espacio de kernel (con los mismos controladores y módulos de kernel) que el sistema operativo *host*, pero cada contenedor tiene su propio espacio de usuario aislado. Los contenedores LXC aprovechan las capacidades de espacio de nombres y administración de recursos de *cgroups* en Linux, que es cómo los contenedores LXC implementan el control y el aislamiento de recursos.

Pueden implementarse de dos formas diferentes, como aplicación o como contenedores del sistema.

En el año 2009 se incorporó al sistema base de Linux la tecnología de contenedores LXC.

2.2.3 Aparición de Docker

Inicialmente Docker Inc (empresa que desarrolla diversas soluciones software como Docker Hub [10] o Docker Desktop [11]) se denominaba dotCloud. Sus fundadores Kamei Founadi, Solomon Hykes y Sebastien Pahi constituyeron la compañía en la ciudad de Paris, en el año 2.008, siendo trasladada en el año 2.010 a los Estados Unidos [13]. La tecnología Docker incorporó una serie de herramientas que, para su época, resultaban muy novedosas, tales como:

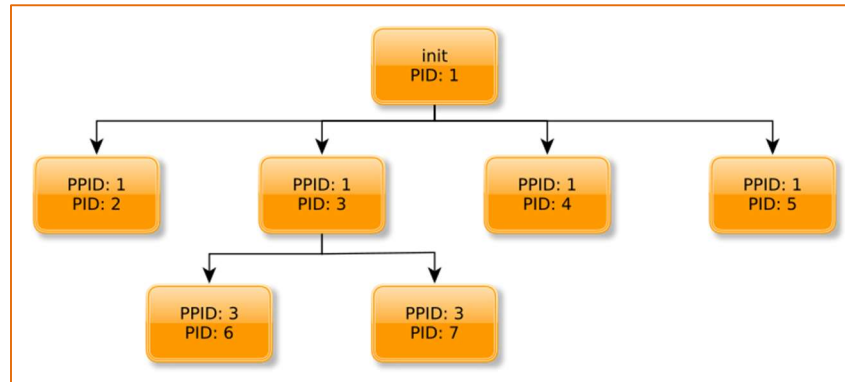
- Interfaz de línea de comandos sencilla para ejecutar y diseñar nuevas imágenes en capas.
- Un demonio de servidor.
- Una biblioteca de imágenes de contenedores previamente diseñadas.
- El concepto de un servidor de registros.

Estas tecnologías combinadas permitieron que los usuarios diseñaran y pusieran en funcionamiento rápidamente nuevos contenedores en capas y los compartieran con otros de una forma dinámica, sencilla y rápida.

En el año 2.015, Microsoft [14] lo agregó a su sistema base, en la rama de los sistemas operativos de servidor, formando parte del sistema Windows Server 2016. Posteriormente fue incorporado al sistema operativo de escritorio Windows 10.

Docker nace de un proyecto de código abierto e incorpora una API de alto nivel que proporciona la administración y utilización de contenedores ligeros que se ejecutan bajo procesos aislados en el sistema operativo. Entre las funcionalidades del *kernel* que hace uso Docker, destacan:

- **CGroups** [15]: característica del *kernel* de Linux que permite que los procesos se organicen en grupos jerárquicos con el fin de limitar y monitorizar el uso de varios tipos de recursos. Cada proceso corre en su propio espacio de *kernel* y de memoria. En concreto, permiten:
 - Establecer límites de recursos: CPU, memoria, I/O, etc.
 - Realizar funciones de auditoría.
 - Priorizar recursos.
 - Etiquetar tráfico de red con un identificador de clase.
 - Congelar grupos de procesos, sus puntos de control y reinicio.
- **Namespaces**. Permiten encapsular recursos globales del sistema de forma aislada. Los cambios de un recurso sólo son visibles para los procesos que se están ejecutando dentro del mismo espacio de nombres, evitando ser utilizados por procesos externos al entorno actual. Existen seis tipos básicos de *namespaces*, que son:
 - **PID** [16]
 - Aísla los procesos que se generan dentro del contenedor, de forma que NO tienen ninguna vinculación con los creados en el servidor anfitrión. El contenedor tendrá su propia jerarquía, generando su propio árbol de procesos desde el proceso padre INIT (PID 1), como puede observarse en el ejemplo de la Figura 2-1.


Figura 2-1 Tabla de Procesos kernel Linux¹

○ NETWORK

- Todos los recursos de red empleados dentro del controlador (interfaces, direcciones, puertos, tablas de ruta, etc.) estarán encerradas en el propio contenedor y no tendrán vinculación con los mismos elementos del *host* anfitrión. Esta clausura asegura que estos procesos no podrán realizar conexiones fuera de este *namespace*, incluso si el proceso es comprometido a través de alguna vulnerabilidad de seguridad.

○ UTS [17]

- Permite que el proceso, dentro del nodo del *UTS namespace*, tenga su propio:
 - Nombre del nodo
 - Nombre del dominio NIS
- Estos dos elementos pueden ser consultados mediante la llamada al sistema *uname()* y *gethostname()*, y pueden ser configurados con el comando *sethostname()*.

○ MOUNT

- Este entorno fue el desencadenante para la creación de la primera jaula “*chroot*”. Su misión es aislar los puntos de montaje del sistema de archivos que pueden ser utilizados por un grupo de procesos. Con esta característica, los procesos en diferentes *namespaces* pueden utilizar distintas vistas de la jerarquía del sistema de archivos. Esto nos permite tener una raíz (“/”) diferente para cada proceso aislado.

○ USER

- Enjaula los identificadores de grupos y usuarios para cada *namespace*. Los ID de usuario y grupo de un contenedor pueden ser diferentes dentro y fuera del *namespace* de usuario. Un ejemplo de ello es que un proceso puede tener un ID de usuario normal, sin privilegios, fuera del *namespace* de usuario y al mismo tiempo tener un ID de usuario 0 (*root*) dentro del contenedor

○ IPC [18]

- Utilizado para aislar objetos *System V* y colas de mensajería *POSIX*.
- Cada espacio de nombres de IPC tiene su propio conjunto de identificadores de *System V IPC* y su propio sistema de archivos de cola de mensajes *POSIX*. Los objetos creados en un espacio de nombres IPC son visibles para todos los demás procesos que están dentro del mismo espacio de nombres, pero no son visibles para los procesos en otros espacios de nombres de IPC.

¹ <https://clibre.io/blog/por-secciones/hardening/item/384-namespaces-aislar-los-procesos-de-linux-en-sus-propios-entornos-de-sistema>

A continuación, en la Figura 2-2, se puede observar y verificar cómo los procesos que se ejecutan dentro de un contenedor tienen *namespaces* distintos, lo que viene a afirmar que se ejecutan en espacios aislados e independientes entre sí. En el primer escenario podemos visualizar la ejecución de una *Shell* dentro de un contenedor utilizando Docker. El PID del contenedor es el 145 desde el punto de vista del sistema anfitrión, que corresponde con el proceso 1 dentro del contenedor y que identifica el proceso *sh*, el primer comando ejecutado en el contenedor.

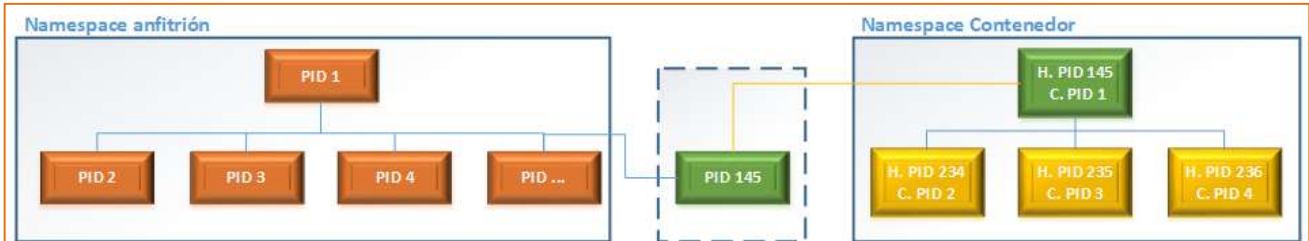


Figura 2-2 Procesos aislados en namespaces independientes. Autoría propia.

Una gran fortaleza de Docker reside en su facilidad para que los usuarios puedan confeccionar, construir y compartir sus propios contenedores en forma de imágenes. Una imagen se define como una plantilla, de solo lectura, que incluye cualquier definición para cualquier biblioteca o dependencia que el código necesite para su ejecución. Es un fichero comprimido (*tarball*) que es instanciada (en ejecución).

Cada imagen está compuesta por una serie de capas que contienen todo el sistema de ficheros inicial en los que se va a basar el contenedor para su funcionamiento, así como su punto de entrada o *entrypoint*. Este punto de entrada se refiere a la aplicación o comando que deberá ejecutarse una vez que se lance la ejecución del contenedor, en el sistema anfitrión, que esté asociado a esa imagen de Docker.

De esta forma, las imágenes se encargan de actuar como un “script” o conjunto de instrucciones útiles encargadas de construir un contenedor, así como una plantilla. Para construir una nueva imagen, basta con seleccionar una “imagen base”, hacer las modificaciones que se consideren oportunas y, finalmente, publicar y compartir el nuevo contenedor configurado.

Docker dispone de su propia base de plantillas, conocida como “*Docker Hub*” [19], que viene a ser un repositorio donde los usuarios registrados pueden almacenar, compartir, descargar y consultar las imágenes Docker creadas en la comunidad.

Además, cada usuario puede crearse su propio repositorio “privado” donde poder almacenar y gestionar, tanto las plantillas descargadas de Docker Hub, como las generadas a partir de plantillas base o creadas por él mismo.

2.3 Diferencias entre los distintos tipos de contenedores

- Almacenamiento
 - Mientras que LXC permiten almacenar los datos de la aplicación en el mismo contenedor, en Docker es necesario definir una ubicación externa (volumen)
- Ejecución de acceso a los recursos
 - Un contenedor LXC es más liviano y consume menos recursos que otro desarrollado con la tecnología Docker.
 - LXC, a través de su API REST de red, permite automatizar complementemente el proceso de implementación y administración de múltiples contenedores en un mismo Sistema.
- Virtualización
 - LXC proporciona una virtualización completa del Sistema, mientras que Docker solo permite la virtualización de aplicaciones.
- Plataforma

- LXC solo es compatible con la plataforma Linux, mientras que Docker puede ejecutarse en sistemas Linux, Windows y Mac.

2.4 Orquestación de contenedores

2.4.1 Introducción

La orquestación de contenedores es una metodología que ofrece una visión superior de la gestión individual de los contenedores, proporcionando visibilidad y control sobre dónde se implementan y cómo se asignan las cargas de trabajo de múltiples contenedores. Esta técnica es esencial para poder gestionar, escalar y ejecutar un grupo de contenedores en numerosos nodos informáticos al mismo tiempo; además, el administrador del sistema podrá aplicar políticas de manera selectiva u holística a una colección de ellos.

En resumen, la orquestación de contenedores ofrece diversos beneficios como los siguientes:

- Reducir la complejidad operacional en la gestión de contenedores.
- Mejorar la seguridad, reduciendo los errores humanos gracias a la automatización de tareas.
- Permitir escalar y reiniciar contenedores y *clusters* automáticamente.
- Ayudar a los equipos de TI a planificar tareas y automatizar cargas de trabajo.

2.4.2 Docker Bundle

En las versiones previas a la 1.12, para poder utilizar estos contenedores se ofrecía, como servicio de orquestación, la plataforma Docker Swarm [20], con la que se podía configurar un *pool* de equipos, que actuaban como anfitriones de los contenedores Docker, con los que cualquier usuario lograría administrar su propia infraestructura.

A partir de la versión 1.12, Docker introduce un nuevo concepto denominado **Distributed Application Bundle** [21] (también conocido como modo enjambre), en el cual los contenedores y nodos forman parte de una red descentralizada, como se observa en la Figura 2-3, donde cada uno de ellos trabaja de forma independiente. Un Docker Bundle es una declaración definida de un conjunto de servicios que especifica:

- Qué revisión de imagen concreta hay que desplegar.
- Qué redes crear.
- Cómo deben conectarse en red los contenedores con sus servicios para poder ejecutarse.

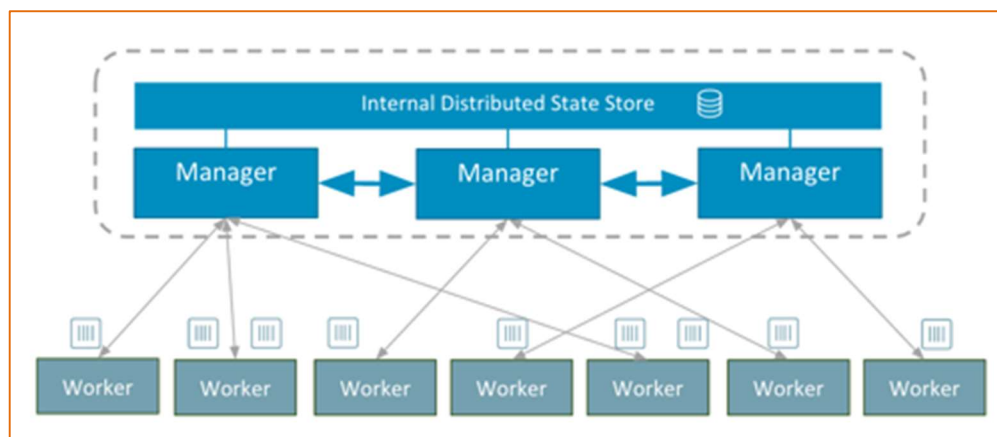


Figura 2-3 Orquestación en Docker. ²

² <https://clouddayscom.files.wordpress.com/2017/10/q5zcsver06ouoljzvfbsvrntzqrqzn7y9abztaxoq8q.png>

2.4.3 Kubernetes

Kubernetes (habitualmente referido también como *short k8s*), software de código abierto, surge como idea en otoño del año 2013 de mano de tres desarrolladores de Google (Craig McLuckie, Joe Beda y Brendan Burns), y tiene su punto de arranque en el año 2014. Es una herramienta portable, extensible, de código abierto, que administra cargas de trabajo de contenedores y servicios, facilita, tanto la configuración como la automatización de un ecosistema para que pueda crecer de forma rápida y sostenible.

Como características más importantes destacan: [22]

- **Servicio de control del tráfico y balanceo de carga:** puede exponer un contenedor usando la resolución DNS del contenedor o bien utilizando la IP del mismo (de forma directa). Si Kubernetes detecta que el tráfico hacia un contenedor es alto, puede equilibrar la carga de peticiones y distribuirlas a otros contenedores de la red para que esta carga permanezca estable.
- **Orquestación de almacenamiento:** habilita la posibilidad de montar un sistema de almacenamiento a elección del usuario, como puede ser un sistema local o un sistema basado en la nube (pública, privada, etc.)
- **Implementaciones y restituciones automatizadas:** a la hora de desplegar nuevos contenedores, el administrador puede indicar el estado en que desea que se encuentren y posteriormente puede ir modificando dicho estado a otros tipos de ejecución. Pueden realizar tareas automáticas que creen nuevos contenedores, que los eliminen o que cambien los recursos de los que se encuentran desplegados.
- **Empaquetado y despliegue optimizado:** permite aprovechar al máximo los recursos hardware de los sistemas anfitrión donde desplegar contenedores. Basta con indicar a Kubernetes cuánta memoria RAM necesita un contenedor y cuántos recursos de CPU para que el sistema pueda ubicar y desplegar el contenedor en el equipo adecuado.
- **Reparación automática:** reinicia los contenedores con problemas y fallos de ejecución, reemplaza y elimina aquellos que no responden a las llamadas de verificación del sistema central definido por el administrador, y los retira del entorno de ejecución mientras no están en disposición de ser utilizados de nuevo.
- **Seguridad:** facilita el almacenamiento seguro de contraseñas, *tokens* OAuth y claves SSH que el administrador utiliza en el despliegue de contenedores, sin exponerlos en los ficheros de configuración.
- **Despliegue en múltiples plataformas:** una de las ventajas más destacadas es la posibilidad de utilizarlo tanto en sistemas locales (ordenadores personales), como en servidores con sistema operativo Linux, Windows o Mac, así como en sistemas en la nube (pública o privada).

Por todas estas razones, Kubernetes se ha convertido en un estándar, y en una de las opciones más viables, para desplegar una infraestructura de contenedores o aplicaciones de contenedor, tanto en servidores físicos como en una nube.

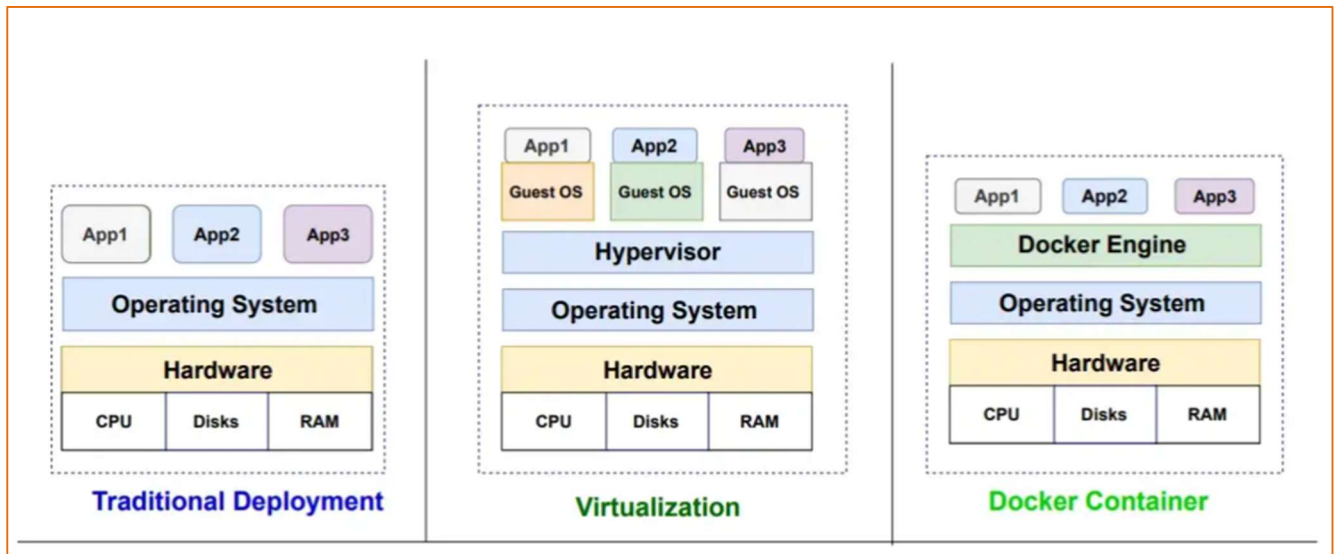


Figura 2-4. Tipos de despliegues de aplicaciones³

En la Figura 2-4 podemos observar la evolución y los componentes que intervienen en los sistemas tradicionales, los sistemas virtualizados y los contenedores Docker.

La arquitectura y funcionamiento de Kubernetes se basa en nodos, los cuales contienen los denominados *pods*, encargados estos últimos de albergar uno o más contenedores que procesan un trabajo o ejecutan un servicio.

Los nodos se dividen en dos tipos principales [23]:

- Nodo **master**: encargado de administrar, configurar y controlar el *cluster*. Monitoriza todos los nodos y *pods* del *cluster* y controla las aplicaciones e imágenes desplegadas. Para ello, ejecuta los siguientes procesos [24]:
 - *etcd*: base de datos, tipo clave/valor, con la que mantiene la configuración global del *cluster*.
 - *kube-scheduler*: decide en qué nodo se ejecuta un determinado contenedor.
 - *kube-controller-manager*: responsable de detectar y responder cuando un nodo deja de funcionar. Encargado de mantener el número correcto de *pods* para cada controlador de replicación del sistema.
 - *kube-apiserver*: expone la API de Kubernetes para que el administrador del *cluster* pueda ejecutar acciones tales como: desplegar contenedores y aplicaciones, obtener una lista de nodos desplegados, estado de carga de los nodos, etc. Estas acciones suelen llevarse a cabo, generalmente, mediante la herramienta CLI (*Command Line Interface*) denominada **kubectl**.
- Nodos **worker**: encargados de ejecutar los contenedores y aplicaciones en *pods*. Ejecutan los siguientes procesos:
 - *kubelet*: encargado de observar los contenedores desplegados con Kubernetes, asegurándose de que están funcionando de forma adecuada; es decir, gestiona el ciclo de vida de los *pods*.

³ https://miro.medium.com/max/720/1*bRDTkA3-l-3vqN3rr_AjtQ.webp

- *kube-proxy*: gestiona las reglas de red para la comunicación con los *pods*, ya sea dentro del *cluster* o desde el exterior.

Al mismo tiempo, *Kubernetes* utiliza *plug-ins* de red para la comunicación entre contenedores, llamados *CNI* (*Container Network Interface*). Estos *plug-ins* definen las librerías y especificaciones necesarias para configurar las interfaces de red de los contenedores desplegados. Los *CNI* más utilizados son Flannel [12] y Calico [13].

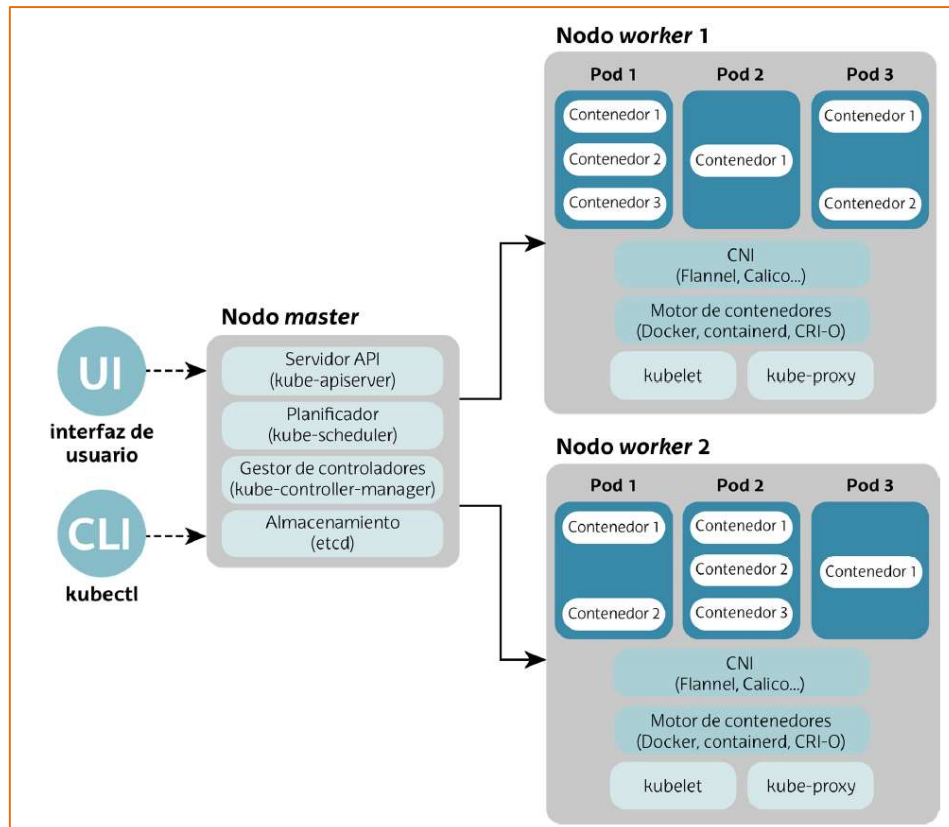


Figura 2-5. Arquitectura de un cluster Kubernetes. Autoría propia.

2.5 Herramientas para la orquestación de *Kubernetes*

2.5.1 Red Hat OpenShift

Otro orquestador disponible en el mercado es Red Hat OpenShift Container Platform [27], comercializado por la empresa Red Hat, la cual es ampliamente conocida en el mundo *open source* por su distribución Linux y sus herramientas empresariales.

Este software es un orquestador, cuya base operacional está basada en *Kubernetes*, capaz de desplegar contenedores empresariales rápidamente tanto en una nube privada como en una nube híbrida. Incluye la distribución Linux empresarial, gestiona el tiempo de ejecución de contenedores, conexiones de red, supervisión, registro de contenedores, autenticación y soluciones de autorización.

Se ha definido como un “Kubernetes empresarial”, dado que han integrado en la solución *open source* otras capas software que complementan y enriquecen el software base Kubernetes. Los distintos módulos y elementos que componen esta solución pueden observarse en la Figura 2-6.

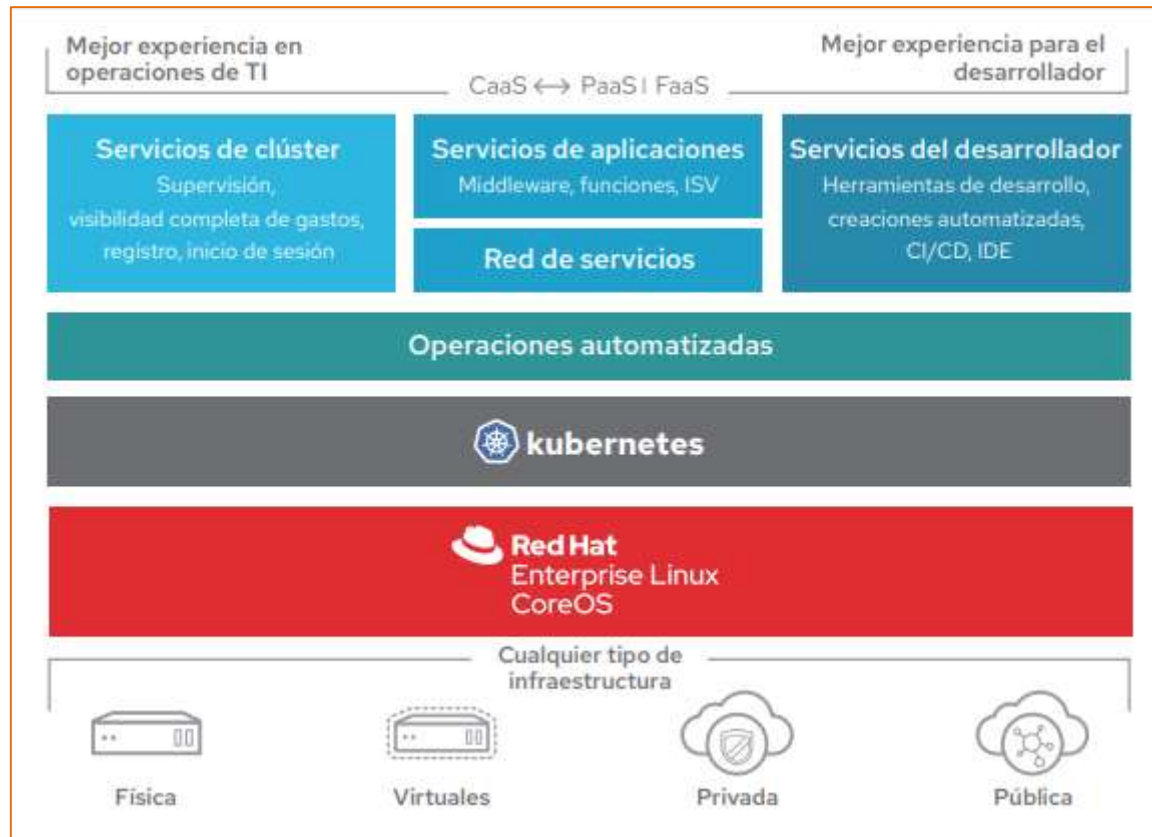


Figura 2-6. Descripción general de la arquitectura Red Hat OpenShift 4. (tomada de [27])

Entre las características más importantes de esta herramienta, destacan las siguientes:

Característica	Ventaja
Escalabilidad	Las aplicaciones que se ejecutan en OpenShift Container Platform pueden expandirse a miles de instancias en cientos de nodos en cuestión de segundos y con facilidad.
Portabilidad de los contenedores	Las imágenes de contenedores creadas según el estándar del sector de la Open Container Initiative (OCI) garantiza la portabilidad entre las estaciones de trabajo del desarrollador y los entornos de producción de OpenShift Container Platform.
Almacenamiento persistente	OpenShift Container Platform admite el almacenamiento persistente, así que permite que los usuarios ejecuten aplicaciones con estado y aplicaciones sin estado nativas de la nube
Estándares <i>open source</i>	OpenShift Container Platform incorpora los contenedores formateados de Docker o según la OCI y Kubernetes con certificación de la Cloud Native Computing Foundation (CNCF) para la organización de los contenedores, además de otras tecnologías de open source.

Tabla 2-1 Características Red Hat OpenShift (tomada de [27])

2.5.2 Kubectl

Kubectl [28] es la interfaz de línea de comandos diseñada para controlar despliegues clusterizados de Kubernetes. Es la forma estándar de comunicación con el *cluster*, permitiendo llevar a cabo todo tipo de operaciones sobre el mismo.

Entre las operaciones que pueden llevarse a cabo en el *cluster*, figuran las siguientes: arrancar, controlar, inspeccionar, gestionar, desplegar y escalar aplicaciones.

Para poder trabajar con este software, en un sistema operativo Linux con arquitectura del procesador de 64 bits, basta descargarlo desde su repositorio oficial, e instalarlo en una ruta de acceso, con permisos de ejecución.

2.5.3 Otros administradores de cluster para Kubernetes

Dentro del ecosistema de orquestadores para Kubernetes, podemos encontrar los siguientes:

- **Nomad** [29]: herramienta de orquestación de pequeñas cargas de trabajo, desarrollada en código abierto, diseñada para desplegar y gestionar contenedores y aplicaciones no contenerizadas.
 - Es una pequeña pieza de software que se instala en los servidores que forman parte de un *cluster*. Se compone de dos piezas:
 - Parte servidor. Gestiona los despliegues
 - Parte cliente. Aloja los despliegues
- **Minikube** [30]: definida como una distribución reducida de Kubernetes, se trata de una buena herramienta para iniciarse en este tipo de gestión, aunque no es la mejor opción desde el punto de vista de las funcionalidades que ofrece. Para su uso es necesario disponer de una herramienta de virtualización (Virtualbox) y activar la característica de virtualización del procesador.
- **Apache Mesos** [31]: Plataforma de orquestación que puede ejecutarse en sistemas operativos Linux, Windows y Mac, concebida para orquestar y gestionar *clusters* de varios nodos de trabajo. Inicialmente, el proyecto fue desarrollado por la Universidad de Berkeley (California) y, posteriormente, pasó a ser desarrollado por la fundación Apache Software. Se compone de los siguientes elementos:
 - Tres nodos *maestros*: Uno *activo* y dos en *standby*.
 - Proceso *master*, que se ejecuta en los tres servidores maestros en el denominado *ZooKeeper Quorum*.
 - Varios nodos tipo *framework*, encargados de ejecutar las tareas definidas por los nodos maestros.
 - Articulado en dos procesos:
 - *Scheduler*: se comunica con los nodos maestros para indicarles los recursos que tiene disponibles y aceptar las peticiones de estos para llevar a cabo las cargas de trabajo.
 - *Executor*: procesa las tareas enviadas por los nodos maestros.
- **Kind** [32]: software diseñado para trabajar, de forma local, con la tecnología Kubernetes y contenedores Docker. Originalmente se desarrolló como herramienta para llevar a cabo pruebas de integración en Kubernetes, simulando su ejecución en un entorno local. Puede instalarse en sistemas operativos Linux, Windows y Mac.
- **Kubespray** [33]: conjunto de *scripts*, desarrollados bajo Ansible, que realizan el aprovisionamiento, configuración y despliegue de contenedores Docker. Como Ansible utiliza conexiones bajo el protocolo SSH, es necesario configurar el acceso mediante clave pública/privada en todos los nodos que formen parte del *cluster* de producción, de

forma que pueda conectarse con los servidores y escribir un inventario con los roles de dichos sistemas. Su mayor ventaja es la posibilidad de desplegar, configurar y actualizar contenedores Docker en cualquier sistema operativo destino, como puede ser Linux, Windows o Mac.

2.6 Software para la administración y automatización de procesos. Ansible

Permite la configuración de múltiples nodos conectándose a ellos utilizando el protocolo SSH y, mediante la ejecución de *scripts* de Python, aplicar estas configuraciones. El formato de los ficheros que se definen en Ansible, denominados *playbooks*, utilizan el lenguaje YAML.

La ventaja de esta herramienta es la facilidad que ofrece a los administradores para aplicar diversas configuraciones en multitud de sistemas e informar del estado final de su ejecución. Funciona mediante módulos en los que poder estructurar los comandos y configuraciones que se pretenden desplegar.

Componentes de Ansible [34]:

- **Modules:** son las librerías que se utilizan para controlar servicios, ficheros, paquetes o comandos.
- **Inventory:** fichero donde se definen los *hosts* o grupos de *hosts* a los que aplicar las configuraciones programadas.
- **Roles:** tareas que pueden ejecutarse en un grupo de *hosts*.
- **Files:** carpeta donde se almacenan los ficheros que deben ser copiados en los *hosts* definidos en los *playbooks*.
- **Task:** carpeta donde se almacenan los ficheros *playbooks*, con la estructura YAML, para ser ejecutados en los *hosts* que pertenecen a un rol determinado.

2.7 Desarrollo de la aplicación web

Si bien la web está en continua evolución, y los sitios y páginas web actuales se asemejan a las tradicionales aplicaciones de escritorio, con la aparición de nuevos *frameworks* de desarrollo como pueden ser Node.js [35] o React [36], que ofrecen una mayor velocidad y fluidez en la interacción con el usuario, todavía siguen persistiendo tecnologías y software que son utilizados habitualmente para el desarrollo de aplicaciones web.

Actualmente la web se caracteriza por una mayor utilización de los dispositivos móviles y sistemas *back-end* denominados *cloud* y evoluciona a su versión 4.0 (web activa), que se fundamentará en estos cuatro pilares:

- **Técnicas *speech-to-text*** y comprensión del lenguaje natural. La nueva web podrá comprender el lenguaje humano y podrá expresar la información escrita y hablada, utilizando una perfecta semántica.
- **Nuevos modelos *M2M*** (comunicación máquina-máquina). Los agentes inteligentes podrán comunicarse entre sí, intercambiando información.
- **Análisis del sentimiento** o uso de la información de contexto. Son dispositivos denominados *wearables*, que monitorizarán determinados aspectos del ser humano, como por ejemplo la tensión arterial, temperatura corporal, ritmo cardíaco, etc.
- **Interacción con el usuario.** La web será capaz de ejecutar acciones con las que dar respuesta a las necesidades de los usuarios.

Mientras todas estas nuevas tecnologías avanzan, y se incorporan a nuestra sociedad, los sistemas actuales basados en webs tradicionales son capaces de proporcionar al usuario todas las características mencionadas. La mayor parte de aplicaciones web actuales están basadas en la arquitectura *MVC*

(Modelo, Vista y Controlador), como se puede observar en la Figura 2-7, dividiendo la aplicación en tres capas:

- **Modelo**
 - Contiene una representación de los datos que maneja el sistema, sus mecánicas de persistencia y su lógica de negocio.
 - Permite el acceso a la capa de almacenamiento de datos.
 - Especifica las reglas de negocio.
 - Administra un registro de controladores y vistas del sistema.
- **Vista**
 - También conocida como interfaz de usuario, es la encargada de representar la información que se envía al cliente y utilizar mecanismos de interacción con éste.
 - Reciben los datos del modelo y los muestra al usuario.
- **Controlador**
 - Intermediario entre la Vista y el Modelo, gestiona el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.
 - Es la inteligencia del sistema y de la aplicación.

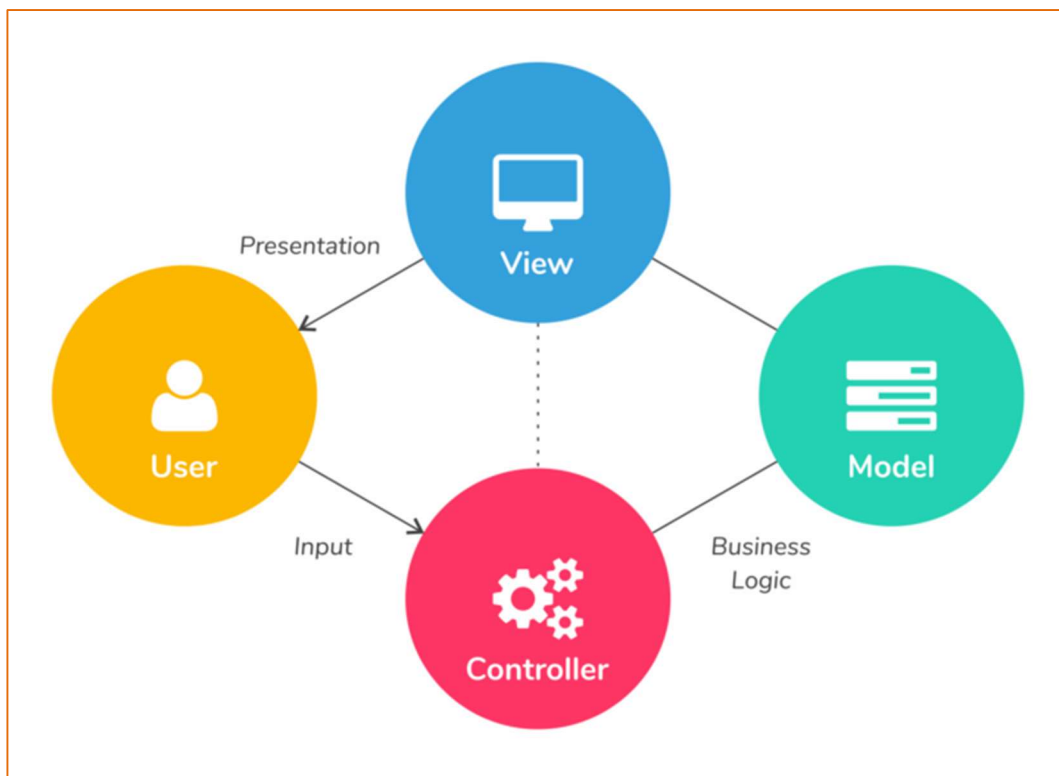


Figura 2-7. Modelo MVC⁴

Para realizar aplicaciones utilizando esta tecnología, podemos hacer uso de una gran variedad de *frameworks*, especialmente para el componente Vista.

2.7.1 Bootstrap

Bootstrap [37] es una biblioteca multiplataforma, especialmente indicada para diseñar e implementar el componente *front-end* o interfaz del usuario (Vista) en aplicaciones basadas en HTML, CSS y JavaScript.

⁴ https://miro.medium.com/max/720/0*9WPgERWsZkDPUpkT.png

Tiene la particularidad que detecta el dispositivo sobre el que se está visualizando la aplicación y se adapta a sus características (PC, portátil, SmartPhone, etc.)

Fue creada por Twitter [38] y originalmente se denominó *Blueprint*, siendo desarrollada por **Mark Otto** y **Jacob Thomson**, como un marco de trabajo para fomentar la consistencia entre las herramientas internas en las que se basaba esta red social. En agosto de 2011, Twitter publicó este software bajo licencia *open source* y, en febrero del 2012, se convirtió en el proyecto de desarrollo más popular de GitHub, siendo actualizado y utilizado hoy en día por multitud de desarrolladores software.

Dispone de una amplia gama de herramientas basadas en código reutilizable, de forma que el desarrollador puede utilizar componentes básicos de una aplicación sin tener que escribir código de diseño. Utiliza jQuery [39] de manera extensiva (*Plug-ins*), proporcionando una amplia compatibilidad con la mayor parte de los navegadores actuales.

Para utilizarlo basta con descargar la hoja de estilo Bootstrap CSS y enlazarla en los ficheros HTML desarrollados. Si se desea hacer uso de los componentes JavaScript, también deberían descargarse o referenciarse junto con la librería jQuery en los archivos HTML de la aplicación.

Actualmente (diciembre de 2.022), la versión disponible para su descarga es la 5.2.2.

2.7.2 PHP

PHP (PHP: *Hypertext Preprocessor*) es un lenguaje de programación, de código abierto, especialmente indicado para su utilización en la creación de sitios web dinámicos e interactivos. Desde sus inicios, en el año 1994, hasta su última versión disponible (8.21 – diciembre 2.022), ha experimentado numerosos cambios y mejoras, lo que lo ha convertido en uno de los lenguajes más populares para el desarrollo de aplicaciones web.

El código PHP se procesa en el lado del servidor (*back-end*), mediante un intérprete PHP previamente instalado, devolviendo la respuesta al navegador web del lado cliente.

Entre sus características, destacan:

- **Documentación y plantillas:** al ser un lenguaje abierto y utilizado por una gran parte de la comunidad, hay publicadas gran cantidad de plantillas y código a disposición de los usuarios.
- **Orientado a objetos:** agiliza el procesamiento y ejecución de las instrucciones, obteniendo un código más limpio y estable.
- **Multiplataforma:** puede ejecutarse en cualquier sistema operativo actual.
- **Multiconector:** integran una gran cantidad de *plug-ins* con los que poder interactuar con cualquier sistema de base de datos, bien sea relacional, documental, multidimensionales, etc.
- **Fácil integración** con HTML, conviviendo ambos en un mismo fichero.

Puesto que estamos hablando de un lenguaje de programación, con el que poder construir páginas dinámicas en las que el usuario debe introducir información, PHP incorpora una serie de funciones con las que llevar a cabo la validación de datos e impedir que se puedan llevar a cabo ataques del tipo XSS (*Cross-Site Scripting*) y otros.

2.7.3 Nginx

Nginx [40] es un *servidor web/proxy inverso* ligero, de alto rendimiento, multiplataforma, que tiene un consumo menor de recursos que otros productos que realizan la misma función. Proporciona un alto rendimiento soportando mayor carga de trabajo y respondiendo con mayor rapidez a las solicitudes de los clientes.

Una de sus principales ventajas, como servidor web, es velocidad de respuesta que ofrece al servir contenido estático, lo que lo convierte en una excelente opción para funcionar como *proxy inverso* o como balanceador de carga para otros servidores web, como puede ser Apache, optimizando la entrega de contenidos.

Su funcionamiento se basa en eventos, donde cada petición que se realiza por parte del cliente se administra utilizando un solo *thread* o hilo, lo que permite consumir muy pocos recursos, especialmente en memoria.

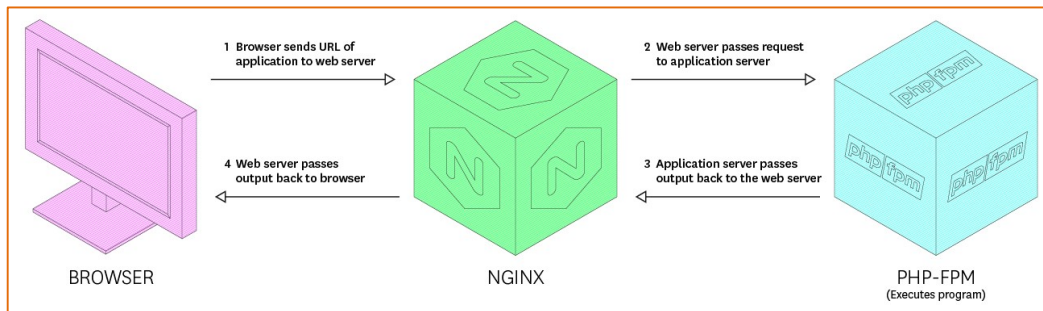


Figura 2-8. Funcionamiento NGINX con PHP-FPM.⁵

Para el procesamiento de páginas con contenido PHP, Nginx utiliza PHP-FPM [41] (*PHP FastCGI Process Manager*), que es una implementación del lenguaje PHP, con algunas características adicionales diseñadas para sitios que tienen una alta carga de peticiones. Como se puede observar en la Figura 2-8, Nginx envía los ficheros PHP al módulo PHP-FPM, el cual procesa el código y devuelve el resultado al servidor web, quien a su vez lo sirve al cliente. La comunicación entre Nginx y PHP-FPM se realiza mediante *sockets* y procesos externos, lo que aísla ambos entornos y ofrece un alto rendimiento incluso al procesar código complejo.

2.7.4 MariaDB

MariaDB [42] es un sistema de gestión de bases de datos, descendiente de MySQL, bajo licencia GPL v2 (General Public License). Su creador, Michael Widenius, desarrolló este software, para el mundo *open source*, debido a que MySQL (su predecesora), que pertenecía a la empresa SUN Microsystems, fue adquirida por Oracle, la cual empezó a licenciar la base de datos MySQL, dejando de ser un producto libre y a disposición de la comunidad.

Puede utilizarse para procesar datos de transacciones de alta disponibilidad, análisis de datos, como servidor integrado, etc. Sigue manteniendo altos niveles de compatibilidad con MySQL, para que los usuarios que estaban utilizando esta última base de datos puedan migrar a la versión gratuita de MariaDB.

Incorpora numerosos mecanismos de almacenamiento, destacando, entre otros *MyISAM*, *Blackhole*, *CSV*, *Memory* y *Archive*.

Gracias a la utilización del motor *aria*, que utiliza la caché para almacenar las filas de datos en lugar de escribirlos a disco, es muy rápida en el procesamiento de consultas complejas.

2.8 Selección y justificación del entorno hardware / virtual / software

Una vez descritos los distintos sistemas y herramientas de las que podemos hacer uso para llevar a cabo este proyecto, gracias a la exhaustiva investigación de las diversas tecnologías planteadas y su respectivo comportamiento, hemos optado por los siguientes productos para llevar a cabo su implantación:

- **Linux:** si bien podríamos haber utilizado Microsoft Windows, por su ligereza, facilidad de configuración y simplificación en la instalación de paquetería Linux es el sistema operativo elegido. Además, mediante la utilización del software Ansible, podremos llevar

⁵ <https://geekflare.com/wp-content/uploads/2018/10/php-and-nginx.png>

a cabo la automatización de configuraciones e instalaciones en todos los nodos del proyecto desde un punto centralizado.

- **Kubernetes:** constituye la herramienta más importante y con la que poder llevar a cabo la orquestación de los distintos contenedores desplegados.
- **Docker:** contenedor de mayor utilización en la comunidad, con mayor proyección, despliegue, repositorio y facilidad de orquestación.
- **Ansible:** software de automatización de tareas especialmente indicado para los sistemas de integración e implementación continua.
- **Bootstrap:** es un *framework* muy intuitivo y fácil de utilizar, poniendo a disposición del desarrollador código de multitud de elementos con los que ahorrar tiempo en el diseño de la aplicación. Utiliza una cuadrícula con la que ubicar fácilmente los distintos componentes de la interfaz de usuario. Permite su utilización con lenguajes de lado de servidor como ASP.NET, PHP e incluso Ruby. Además, es altamente personalizable, incluye componentes prediseñados y está respaldado por una gran comunidad que lo mantiene y desarrolla nuevas versiones.
- **PHP:** es un lenguaje abierto, portable, estándar, multiplataforma, cuyo desarrollo puede procesarse en cualquier sistema operativo. De fácil configuración y programación, existe una gran cantidad de código reutilizable, a disposición de la comunidad. Se integra perfectamente con cualquier servidor web del mercado y con la base de datos MariaDB. El código de una aplicación puede ser reutilizado en otras, lo que supone un ahorro de tiempo considerable.
- **Nginx con el módulo PHP-FPM:** por su escalabilidad, rendimiento y asincronía en la gestión de peticiones, además de su perfecta integración con el intérprete PHP-FPM, con los que aprovechar al máximo los recursos del sistema. Estos dos sistemas, conjuntamente con Linux y MariaDB, reciben el nombre de LEMP (Linux, Nginx (E), MariaDB o MySQL y PHP).
- **MariaDB:** Se integra muy fácilmente con las páginas dinámicas desarrolladas con PHP y tiene multitud de conectores para enlazar con servidores web como Apache, Nginx, Microsoft IIS, etc.

3 DESARROLLO DEL TFM

3.1 Diseño de la infraestructura

3.1.1 Hardware y Software Base

Para la instalación de la infraestructura virtual del proyecto, se ha utilizado, como base de toda la plataforma, un servidor HPE, modelo Proliant DL 160 Gen9 (Figura 3-1), con las siguientes características:

- 2 procesadores Intel® Xeon® ES-2630 v3 @ 2.40GHz
 - 8/8 cores, 16 threads 64 bits
- 32 GB memoria RAM
- Adaptador Ethernet con dos puertos RJ45 de 1 GB.
- 2 HD, de 1 TB de capacidad, en RAID 1



Figura 3-1. Servidor HPE Proliant DL160 – Gen9.⁶

En cuanto al Sistema Operativo base, este servidor incorpora Windows Server 2012, versión DataCenter, especialmente indicado para ejecutar altas cargas de trabajo TI y sistemas de virtualización avanzados. Permite la implantación, monitorización y ejecución de múltiples máquinas virtuales bajo el software Hyper-V. Sobre esta plataforma se instalarán y configurarán las VMs, con sistema operativo Linux (Ubuntu 22.04.5 server AMD 64 LTS), que forman parte de la arquitectura del sistema.

En cuanto al almacenamiento, dado que está previsto que todos los sistemas accedan a un repositorio común en red, dispondremos de un NAS (*Network Attached Storage*), desde el que se exportará un volumen NFS (*Network File System*) al resto de componentes de la plataforma. En este caso se trata de un Synology DS420+, con la siguiente configuración:

⁶https://techlibrary.hpe.com/docs/enterprise/servers/DL160Gen9/DL160Gen9-setup/system_setup_overview/189580.png

- 2 procesadores Intel® Celeron® @ 2.0GHz
- 2 Gb memoria RAM
- 2 Adaptadores Ethernet Gigabit
- 4 HDD/SSD SATA de 3,5" de 8TB cada uno en RAID 5.

3.1.2 Configuración virtual

Toda la infraestructura informática, que compone este proyecto, se sustenta en un conjunto de máquinas virtuales que se ejecutan en esta plataforma hardware.

El conjunto de VMs está dividido en dos grupos:

- 1 VM de control, denominada *Zelenski*, desde la que componer, ejecutar y procesar los *playbooks* de Ansible para su despliegue en el resto de la infraestructura.
- 3 VMs que componen el *cluster* básico de Kubernetes, denominadas:
 - *Putin* (rol *master*)
 - *Biden* (rol *worker*)
 - *Merkel* (rol *worker*)

Los recursos hardware de la VM *Zelenski* y el S.O. instalado, son los siguientes:

Recursos Hardware				
S.O.	CPUs Virtuales	RAM	Almacenamiento	Arquitectura
Ubuntu 22.04.1 Desktop amd64	3	4 GB	50 GB	64 bits

Tabla 3-1 Recursos Hardware Virtuales y S.O. VM Zelenski

En cuanto a las VMs que componen el *cluster* Kubernetes, los recursos hardware virtuales que se han reservado para cada una de las tres *VMs* que componen esta infraestructura, así como el sistema operativo base de cada una de ellas, son los siguientes:

Recursos Hardware				
S.O.	CPUs Virtuales	RAM	Almacenamiento	Arquitectura
Ubuntu 22.04.5 server amd64 LTS	4	6 GB	50 GB	64 bits

Tabla 3-2 Recursos Hardware Virtuales y S.O. VMs cluster Kubernetes

3.1.3 Arquitectura y conectividad

Todas estas *VMs*, así como su plataforma base, están conectadas a una red Ethernet interna que se ha creado para este proyecto y a la que hemos definido como “laboratorio”. En la Figura 3-2 se pueden observar los distintos elementos desplegados y su conectividad en capa 3.

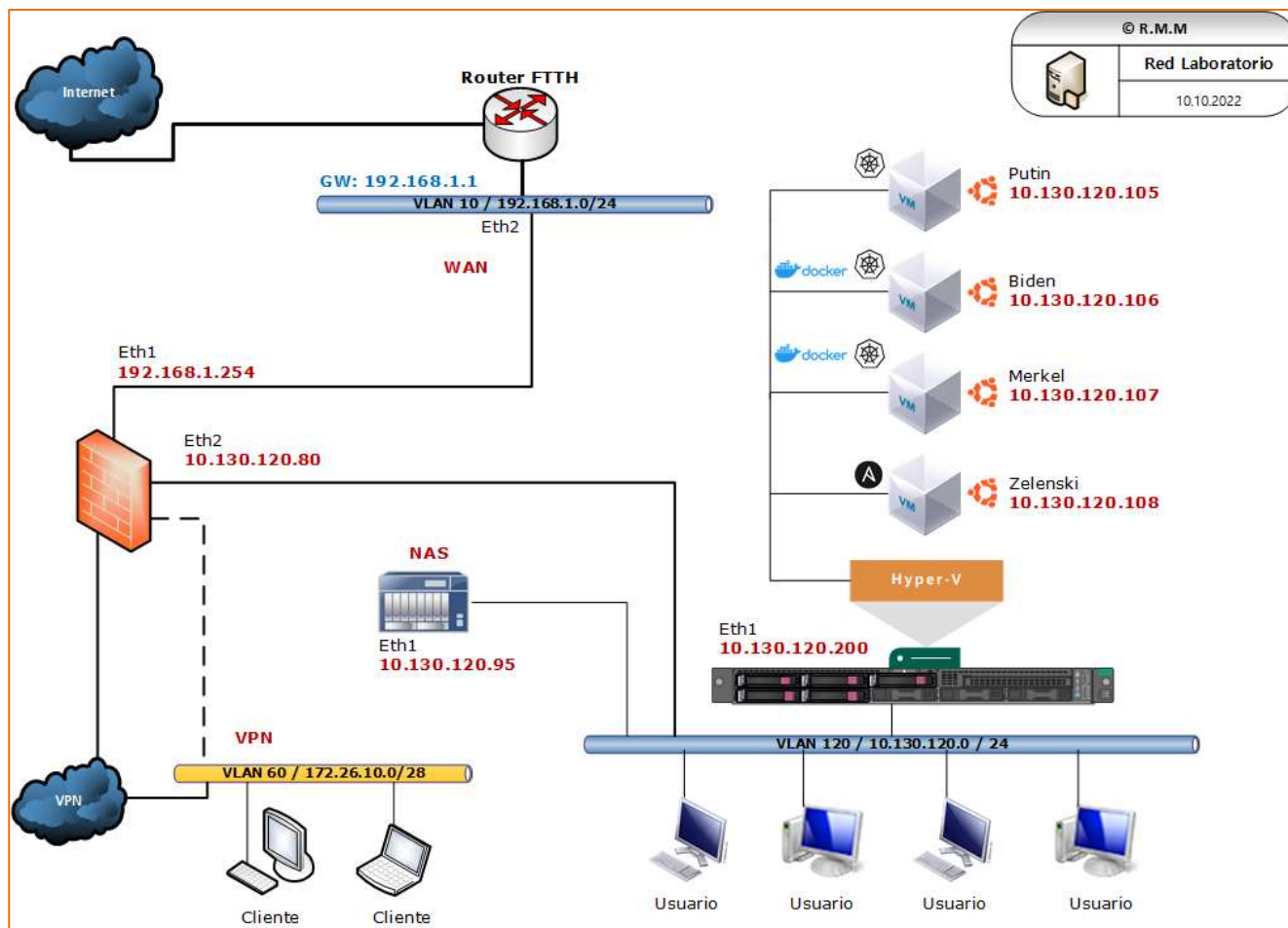


Figura 3-2. Esquema conectividad e infraestructura. Autoría propia.

3.2 Implantación e instalación del clúster Kubernetes

3.2.1 Despliegue del cluster Kubernetes mediante Ansible

Una vez instalados los sistemas operativos base en las cuatro *VMs* que forman parte del proyecto, procedemos con la configuración de los distintos paquetes software necesarios para utilizar Docker y Kubernetes.

En primer lugar, instalamos, en el servidor *zelenski*, el paquete Ansible, que, como ya se comentó anteriormente, es el utilizado para gestionar los distintos nodos del *cluster* mediante conexiones SSH.

Para ello, tecleamos el siguiente comando:

```
root$zelenski:/~# apt-get install ansible -y
```

Completado este primer paso, vamos a generar un par de claves (pública y privada), basadas en criptografía asimétrica, como se muestra en la Figura 3-3, para configurar las conexiones entre el servidor central (*zelenski*) y los servidores que componen el *cluster* Kubernetes (*putin*, *biden* y *merkel*).

El algoritmo seleccionado es RSA (*Rivest, Shamir y Adleman*), con un tamaño de 2.048 bits.

```
root@zelenski:~# ssh-keygen -b 2048 -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa
Your public key has been saved in /root/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:DVpmAjnyEulXYzYKyBdJyyy9sJMspg2T/RU6nUYcTvw root@zelenski
The key's randomart image is:
+---[RSA 2048]---+
| . ++=o          |
|  =+B*..*       |
|   o*=o**=o     |
|  +.=o=oOEo     |
|=.*.+.*.S .     |
|o= o +          |
|. . .           |
|                |
|                |
+---[SHA256]-----+
root@zelenski:~#
```

Figura 3-3. Generando par de claves RSA para conexiones SSH. Autoría propia.

A continuación, es necesario copiar el fichero *id_rsa.pub* en el resto de *hosts* del *cluster* (carpeta */root/.ssh*), para que cuando se inicialice la comunicación, el anfitrión *zelenski* pueda identificarse con su clave privada *id_rsa*. Para ello, nos conectamos a cada uno de los servidores virtuales, nos identificamos como usuario *root* y, en su carpeta */root/.ssh*, volcamos el contenido de este fichero en el archivo *authorized_keys*, encargado de registrar las claves de otros sistemas que permiten su identificación y autenticación.

El resultado final se muestra en la Figura 3-4:

```
root@putin:~/.ssh# ll
total 12
drwx----- 2 root root 4096 oct 31 18:49 ./
drwx----- 9 root root 4096 nov  2 22:07 ../
-rw----- 1 root root 395 oct 31 18:49 authorized_keys
root@putin:~/.ssh# cat authorized_keys
ssh-rsa AAAAB3NzaClyc2EAAAADAQABAAQCIyMctu/in0N9f9usuFn4/TDGge5jguo8a9fntAKIeUiugQZ8CkuIGUTTh
s8QJym3vuuWCpHXOYCN3a5vHgvpAuGzrhf/lbGfsXIp3EjDk68n/eKYWnFg9wOUqbAR/j+run6IvdBNP89Yl2qWRbrqfhM5x
de9YpmA0ym7vIeWmyzXPYK/RTuEkF6kXkc2oyDkxe0YUXrIj8WlvpaKLuMC7zS+spiduA/EllOakcKlHdyj2RzsXvHch7ffH
yPg7bdN4EtsfdImVJUAUNGPTfq1JKB+Q0aUc7XmLT+pTY800FxdR0ny7K6ZB9rdlyyDV7C2SX/huH8DFwrgYAqJkmqgh root
@zelenski
root@putin:~/.ssh#
```

Figura 3-4. Instalación clave pública del servidor “zelenski”. Autoría propia.

En estos momentos, ya estamos en disposición de empezar a instalar Kubernetes, Docker y otro software con dependencia de estos, así como desplegar las configuraciones necesarias para su ejecución en las máquinas virtuales del *cluster*.

En el servidor *zelenski*, creamos una carpeta denominada *k-cluster*.

```
root@zelenski:~# mkdir k-cluster
```

En su interior, vamos a crear el primer fichero de configuración para Ansible, denominado *hosts*, con el siguiente contenido:

```
~/k-cluster/hosts

[masters]
putin ansible_host=10.130.120.105 ansible_user=root

[workers]
biden ansible_host=10.130.120.106 ansible_user=root
merkel ansible_host=10.130.120.107 ansible_user=root

[all:vars]
ansible_python_interpreter=/usr/bin/python3
```

Este fichero es el denominado “inventario” y, como se puede observar, es utilizado para especificar los datos de los servidores de la plataforma, su dirección IP, usuarios remotos de conexión y los grupos en los que están encuadrados. La última línea del fichero de configuración, advierte a Ansible que debe utilizar los intérpretes de Python 3 en los servidores remotos para llevar a cabo las operaciones de administración.

El siguiente paso es la instalación de los paquetes, a nivel del sistema operativo, requeridos por Kubernetes, mediante el administrador de paquetería que utiliza la distribución Ubuntu. Los paquetes que necesitamos instalar son:

- *Docker*: componente principal para alojar y ejecutar los contenedores desplegados.
- *kubeadm*: herramienta CLI con la que poder instalar y configurar los distintos componentes de un *cluster* de manera estándar.
- *kubelet*: servicio del sistema Kubernetes que se ejecuta en todos los nodos y permite la comunicación con el nodo maestro para llevar a cabo las tareas que éste le asigna.
- *kubectrl*: herramienta CLI, que se instala únicamente en el nodo maestro (*putin*), encargada de enviar instrucciones al *cluster* a través de su *API*.

Definimos un nuevo *playbook*, al que denominamos ***dependencias.yml***.

```
~/k-cluster/dependencias.yml

- hosts: all
  become: yes
  tasks:
    - name: Instalación de Docker
      apt:
        name: docker.io
        state: present
        update_cache: true

    - name: Agrega, a los repositorios APT, el repositorio Transport HTTPS
      apt:
        name: apt-transport-https
        state: present

    - name: Agrega la clave del repositorio de Kubernetes
```

```

apt_key:
  url: https://packages.cloud.google.com/apt/doc/apt-key.gpg
  state: present

- name: Agrega el repositorio Kubernetes
  apt_repository:
    repo: deb http://apt.kubernetes.io/ kubernetes-xenial main
    state: present
    filename: 'kubernetes'

- name: Instala el paquete kubelet
  apt:
    name: kubelet=1.25.3-00
    state: present
    update_cache: true

- name: Instala el paquete kubeadm
  apt:
    name: kubeadm=1.25.3-00
    state: present

- hosts: putin
  become: yes
  tasks:
    - name: Instala el programa kubectl
      apt:
        name: kubectl=1.25.3-00
        state: present
        force: yes

```

Como podemos observar, la primera ejecución del *playbook*, lleva a cabo las siguientes tareas:

- Instala Docker.
- Instala *apt-transport-https*, con el que podrá añadir fuentes HTTPS externas a la lista de fuentes APT actuales.
- Agrega la clave *apt* del repositorio de APT de Kubernetes para la verificación de claves y descarga de paquetería.
- Incluye el repositorio de APT de Kubernetes a la lista de fuentes APT de los servidores virtuales del *cluster*.
- Instala los paquetes *kubelet* y *kubeadm*.

El segundo apartado del fichero, realiza una única tarea:

- Instala en el servidor maestro (*putin*), el software *kubectl*.

Ejecutamos el *playbook* con el siguiente comando:

```

root$zelenski:~/kube-cluster# ansible-playbook -i hosts kube-dependencies.yml

```

Si todo se ha procesado correctamente, pasaremos al siguiente apartado de configuración, si bien, antes de llevarlo a cabo, comentaremos algunos aspectos del funcionamiento y configuración de red de un *pod*.

Un *pod* es una unidad atómica que puede ejecutar uno o más contenedores. Estos contenedores comparten recursos, tales como volúmenes de archivos e interfaces de red. Los *pods* son la unidad básica

de programación de Kubernetes y garantizan que todos los contenedores que forman parte de un *pod* deben ejecutarse en el mismo nodo en el que esté programado el *pod*.

Cada *pod* comprende su propia dirección IP, mediante la cual puede comunicarse con otros *pods* ubicados en otros nodos del *cluster*. Esta configuración y comunicación necesita un componente de red, independiente de la interfaz del anfitrión, que pueda realizar comunicaciones con otros *pods* de forma transparente. Como ya se comentó anteriormente, este componente, denominado CNI (*Container Network Interface*), es un *plug-in* de red externo independiente y, los más utilizados son Flannel y Calico. En este proyecto vamos a utilizar Flannel [43], dado que es una opción estable y ampliamente utilizada en ecosistemas Kubernetes.

Llegados a este punto, debemos configurar el nodo definido como maestro (*putin*) y, para ello, confeccionaremos un nuevo *playbook* denominado ***maestro.yml***, el cual, tendrá el siguiente contenido:

```
~/k-cluster/maestro.yml

- hosts: putin
  become: yes
  tasks:
    - name: Inicializa el cluster
      shell: kubeadm init --pod-network-cidr=10.244.0.0/16 >> cluster_initialized.txt
      args:
        chdir: $HOME
        creates: cluster_initialized.txt

    - name: Crea el directorio .kube
      become: yes
      become_user: root
      file:
        path: $HOME/.kube
        state: directory
        mode: 0755

    - name: Copia el fichero admin.conf a la carpeta /root/.kube/config
      copy:
        src: /etc/kubernetes/admin.conf
        dest: /root/.kube/config
        remote_src: yes
        owner: ubuntu

    - name: Instala las redes del Pod
      become: yes
      become_user: root
      shell: kubectl apply -f
https://raw.githubusercontent.com/coreos/flannel/a70459be0084506e4ec919aa1c114638878db11b/Documentation/kube-flannel.yml >> pod_network_setup.txt
      args:
        chdir: $HOME
        creates: pod_network_setup.txt
```

Vamos a desglosar el contenido de este *playbook*, explicando qué se ejecuta en cada uno de sus apartados.

- El primer *play* definido inicializa el *cluster*, ejecutando el comando **kubeadm init**. A este comando le proporcionamos, como parámetro, el argumento:
 - **--pod-network-cidr=10.244.0.0/16**
 - Con ello, le estamos especificando la subred privada que van a utilizar los *Pods* para comunicarse entre ellos y el nodo maestro.
 - La máscara /16 nos va a permitir definir hasta un total de **65.536** direcciones IP.
 - *Flannel* utilizará este direccionamiento por defecto.
 - *kubeadm* utilizará también este direccionamiento de red.
- La segunda tarea, crea un directorio denominado *.kube* en la carpeta raíz del usuario *root*. Esta carpeta es la encargada de almacenar los datos de configuración tales como:
 - Archivos de claves de administrador, necesarias para establecer la comunicación con el *cluster*.
 - Dirección IP y puerto para comunicación con la API del *cluster* (nodo maestro).
- En el tercer paso, copiamos el fichero */etc/kubernetes/admin.conf*, que se ha generado mediante el comando **kubeadm init**, al directorio principal del usuario *root*. Esto permite utilizar el comando **kubectl** para llevar a cabo tareas con el *cluster* recién creado.
- Finalmente, se procesa el comando **kubectl apply** para instalar el CNI Flannel:
 - **kubectl apply -f descriptor.[yaml | json]**
 - Indica a **kubectl** que cree los objetos descritos en el archivo *descriptor.[yaml | json]*
 - El fichero *kube-flannel.yml* contiene las descripciones de los objetos necesarios para configurar Flannel en el *cluster*.

Procesamos este *playbook* ejecutando el siguiente comando:

```
root$zelenski:~/kube-cluster# ansible-playbook -i hosts master.yml
```

Para comprobar que el *playbook* se ha ejecutado correctamente, podemos lanzar el siguiente comando, desde el servidor maestro (*putin*):

```
root$putin:~# kubectl get nodes
```

El resultado de ejecución debería ser como el siguiente:

Output				
NAME	STATUS	ROLES	AGE	VERSION
putin	Ready	master	1m	v1.25.3

Finalmente, configuraremos los nodos tipo *worker* (*biden* y *merkel*), incorporándolos al *cluster* de producción. Para llevar a cabo esta acción, ejecutaremos el comando *kubeadm* con los siguientes parámetros:

- Información para comunicarse con el *cluster* (IP y puerto de la API del nodo maestro).
- *Token* seguro.

- Firma SHA del certificado del nodo maestro para su validación.

El contenido del *playbook* confeccionado, al que hemos denominado *trabajadores.yml*, es el siguiente:

```
~/k-cluster/trabajadores.yml

-- hosts: putin
  become: yes
  gather_facts: false
  tasks:
    - name: Lanzamos el comando get join
      shell: kubeadm token create --print-join-command
      register: join_command_raw

    - name: Recogemos el resultado del comando
      set_fact:
        join_command: "{{ join_command_raw.stdout_lines[0] }}"

- hosts: workers
  become: yes
  tasks:
    - name: Enlazamos con el cluster
      shell: "{{ hostvars['master'].join_command }}" >> node_joined.txt
      args:
        chdir: $HOME
        creates: node_joined.txt
```

Tras procesarse el *playbook*, el resultado conseguido es que los nodos *workers* se han añadido al *cluster* y se encuentran en estado *ready*. Los pasos que ha ejecutado este *script*, son los siguientes:

- El primer *play* ejecutado (**kubeadm token create --print-join-command**), obtiene el comando, con los parámetros necesarios, que debe ser ejecutado en los nodos de trabajo. Lo que realiza realmente, es la ejecución del comando en el servidor *putin* y captura la salida completa del mismo y lo almacena en la variable interna denominada *join_command_raw*.
 - El comando obtenido tiene el siguiente formato:

```
kubeadm join --token <token> <master-ip>:<master-port> --discovery-token-ca-cert-hash sha256:<hash>
```

- El segundo *play* tiene una única tarea: ejecutar este comando en los nodos *workers*.
 - Utiliza la variable *join_command_raw* para añadir los datos del *token*, *dirección* y *puerto* del *master* y finalmente el *hash sha256*.
 - Una vez se haya completado, los dos nodos formarán parte del *cluster*.

Para comprobarlo, ejecutaremos el siguiente comando:

```
root$putin:~# kubectl get nodes
```

El resultado de ejecución debería ser como el siguiente:

Output				
NAME	STATUS	ROLES	AGE	VERSION
putin	Ready	master	3m	v1.25.3
biden	Ready	<none>	1m	v1.25.3
merkel	Ready	<none>	1m	v1.25.3

Como se puede observar en la salida del comando, los nodos *worker* tienen la etiqueta *<none>*. Esto es así porque, por defecto, cuando se añade un nodo al *cluster*, el sistema no los etiqueta por su función. No obstante, podemos cambiar esta información tecleando el siguiente comando:

```
root$putin:~# kubectl label node biden node-role.kubernetes.io/worker=worker
root$putin:~# kubectl label node merkel node-role.kubernetes.io/worker=worker
```

Con estos comandos visualizamos, como etiqueta del nodo, su verdadera función, *worker*.

Output				
NAME	STATUS	ROLES	AGE	VERSION
putin	Ready	master	3m	v1.25.3
biden	Ready	worker	1m	v1.25.3
merkel	Ready	worker	1m	v1.25.3

A partir de estos momentos, estamos en disposición de desplegar aplicaciones o contenedores dentro del *cluster*.

3.2.1 Comprobando el estado del cluster

Para comprobar el estado actual del *cluster*, ejecutaremos varios comandos para chequear su estado. El primero de ellos, cuya salida podemos observar en la Figura 3-5, nos muestra el estado actual del *cluster*:

```
root$putin:~# kubectl cluster-info
```

```
root@putin:~# kubectl cluster-info
Kubernetes control plane is running at https://10.130.120.105:6443
CoreDNS is running at https://10.130.120.105:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
root@putin:~#
```

Figura 3-5. Información del clúster definido en el proyecto. Autoría propia.

Comprobamos, como se observa en la Figura 3-6, que no hay lanzado ningún contenedor y ninguna aplicación en el *cluster*:

```
root$putin:~# kubectl get pods
```

```
root@putin:~# kubectl get pods
No resources found in default namespace.
root@putin:~#
```

Figura 3-6. Información sobre la ejecución de PODS en el *cluster*. Autoría propia.

Y, finalmente, como se muestra en la Figura 3-7 y Figura 3-8, realizamos dos últimas comprobaciones para visualizar el estado de todos los *pods* y servicios desplegados en todos los *Namespaces*.

```
root$putin:~# kubectl get pods --all-namespaces
```

```
root@putin:~# kubectl get pods --all-namespaces
NAMESPACE   NAME                                     READY   STATUS    RESTARTS   AGE
kube-system  coredns-565d847f94-bgjlb              1/1     Running   0           1h
kube-system  coredns-565d847f94-bk6zb              1/1     Running   0           1h
kube-system  etcd-putin                             1/1     Running   0           1h
kube-system  kube-apiserver-putin                   1/1     Running   0           1h
kube-system  kube-controller-manager-putin          1/1     Running   0           1h
kube-system  kube-proxy-7pm9t                       1/1     Running   0           1h
kube-system  kube-proxy-bbkxb                       1/1     Running   0           1h
kube-system  kube-proxy-pjmgp                       1/1     Running   0           1h
kube-system  kube-scheduler-putin                   1/1     Running   0           1h
root@putin:~#
```

Figura 3-7. Información total de los “namespaces” y PODS en el *cluster*. Autoría propia.

```
root$putin:~# kubectl get services --all-namespaces
```

```
root@putin:~# kubectl get services --all-namespaces
NAMESPACE   NAME          TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
default     kubernetes    ClusterIP    10.96.0.1     <none>         443/TCP          3d11h
kube-system  kube-dns      ClusterIP    10.96.0.10    <none>         53/UDP,53/TCP,9153/TCP 3d11h
root@putin:~#
```

Figura 3-8. Información de los servicios actualmente desplegados en el *cluster*. Autoría propia.

3.2.2 Configuración repositorio NFS común al cluster

Para que todos los nodos tengan acceso a un mismo repositorio, donde poder almacenar los datos persistentes, se ha configurado una conexión, vía NFS con el NAS desplegado en la infraestructura del proyecto.

El gráfico de conexión es el indicado en la Figura 3-9.

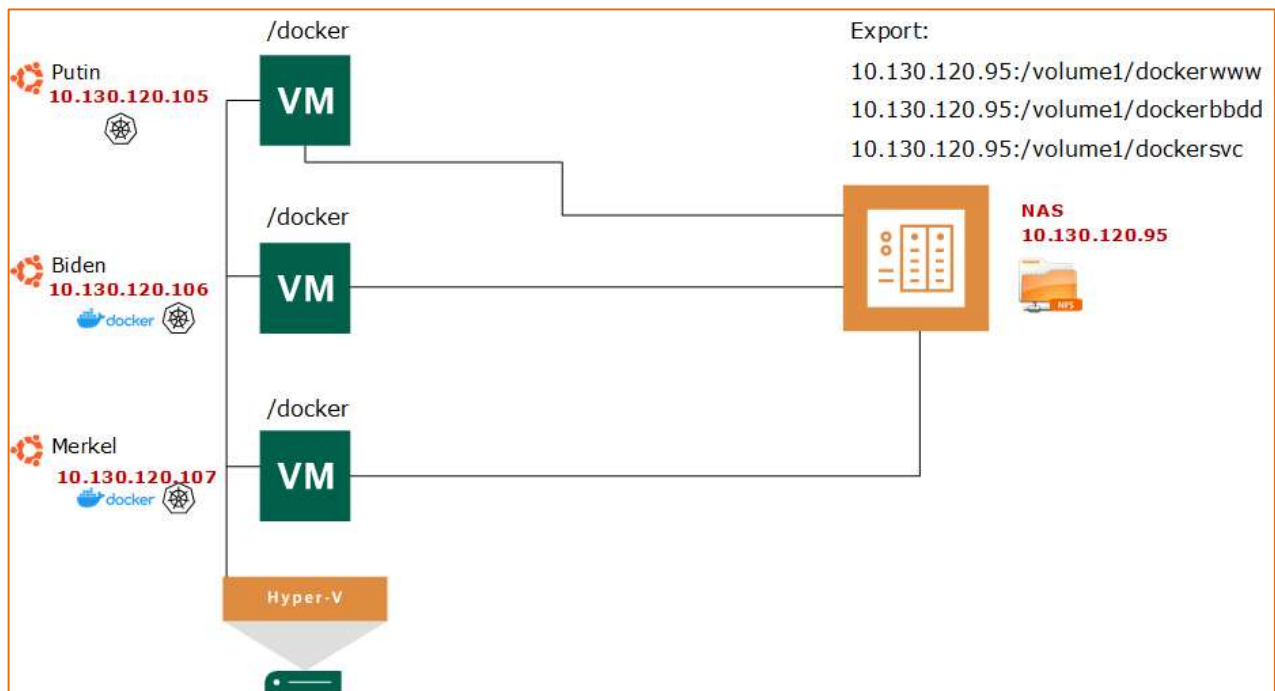


Figura 3-9. Conectividad VMs con repositorio NFS. Autoría propia.

Nos conectamos, vía SSH, a los tres servidores del *cluster*.

Creamos una carpeta, denominada *docker*, en la raíz del sistema en las tres VMs que componen el *cluster*.

```
root$putin:~# mkdir /docker
```

Dentro de esta carpeta, creamos a su vez tres nuevas subcarpetas, que se especifican en la Tabla 3-3.

Origen	Destino	Contenido	Tipo
10.130.120.95:/volume1/dockerwww	/docker/www	Contenedores con servicios web	NFS
10.130.120.95:/volume1/dockerbbdd	/docker/mysql	Contenedores con servicio BD	NFS
10.130.120.95:/volume1/dockersvc	/docker/svc	Contenedores microservicios	NFS

Tabla 3-3 Recursos definidos en los volúmenes “nfs”. Autoría propia.

Instalamos el paquete *nfs-common*, necesario para poder realizar conexiones con este protocolo.

```
root$putin:~# apt-get install nfs-common -y
```

Para que la conexión sea persistente y cada vez que arranquen las VMs, conecten con el repositorio, se ha añadido la configuración al fichero `/etc/fstab`. El resultado se muestra en la Figura 3-10.

```

root@putin:~# cat /etc/fstab
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to name devices
# that works even if disks are added and removed. See fstab(5).
#
# <file system>          <mount point>          <type> <options>          <dump> <pass>
# / was on /dev/ubuntu-vg/ubuntu-lv during curtin installation
/dev/disk/by-id/dm-uuid-LVM-ab7zeloatHe9lwwQ08fg2AVfnlZ0t9lEuEDEjc0zn4MilE2c8ytyGW65zgSTJNsQ / ext4 d    defaults 0 1
# /boot was on /dev/sda2 during curtin installation
/dev/disk/by-uuid/01773506-e949-44c5-b67f-8c77385adcff /boot ext4 defaults 0 1
#/swap.img none swap sw 0 0
10.130.120.95:/volume1/dockerwww /docker/www nfs rw,noexec,auto 0 0
10.130.120.95:/volume1/dockerbdd /docker/mysql nfs rw,noexec,auto 0 0
10.130.120.95:/volume1/dockersvc /docker/svc nfs rw,noexec,auto 0 0

```

Figura 3-10. Contenido fichero *fstab* del servidor *putin*. Autoría propia.

Finalmente, montamos el nuevo repositorio ejecutando el comando:

```
root$putin:~# mount -a
```

3.2.3 Creando volúmenes persistentes

Kubernetes proporciona un sistema de almacenamiento persistente mediante la utilización de volúmenes. Para que no se pierda información cuando un contenedor deja de funcionar, o bien, para poder compartir contenido entre varios contenedores, es necesario la utilización de un volumen común. El objetivo es mantener la información, incluso cuando los *Pods* o contenedores dejen de funcionar. Los volúmenes son considerados como un recurso más en Kubernetes.

Podemos utilizar dos tipos de volúmenes:

- *PersistentVolume* (PV)
 - Son creados por los administradores del sistema y pueden ser aprovisionados mediante las plantillas de configuración de los *Pods*.
- *PersistentVolumeClaim* (PVC)
 - Son solicitudes de almacenamiento que se realizan por parte de un usuario. Es muy similar a lo que es un *pod*. Los *Pods* consumen recursos de *nodos* y los PVC consumen recursos de un PV, es decir, para poder utilizar este almacenamiento dinámico necesitamos contar previamente con un almacenamiento persistente.

Si bien Kubernetes ofrece la posibilidad de utilizar volúmenes en las principales empresas que ofrecen servicios en la nube (AWS [44], Azure [45], OpenStack [46], etc.), los que necesitamos en este proyecto son los denominados *PersistentVolume*.

Un *PersistentVolume* es un volumen que contiene los datos procesados por los contenedores y no son eliminados cuando los contenedores son destruidos. Tiene la propiedad adicional de tener un ciclo de vida independiente de cualquiera de los *Pods* que lo utilicen. Otra de las características de este objeto, es la capacidad de capturar los detalles de la implementación de almacenamiento, que puede ser del tipo NFS, iSCSI o un sistema determinado por el proveedor de la nube y su infraestructura.

En función a los modos de acceso, podemos utilizar las siguientes configuraciones:

- **ReadWriteOnce** (RWO): read-write para un solo nodo
- **ReadOnlyMany** (ROX): read-only para muchos nodos
- **ReadWriteMany** (RWX): read-write para muchos nodos

Y, en función de las políticas de reciclaje, podemos configurar las siguientes:

- **Retain**: retiene el contenido
- **Recycle**: se reutilizará el contenido
- **Delete**: borrará el contenido

Para que los contenedores que ejecuten servicios web, tengan un repositorio común, donde almacenar la configuración e información común a todos ellos, necesitamos crear un repositorio persistente. Utilizando la conectividad con el repositorio NFS anterior, vamos a configurar un fichero *yaml*, con los parámetros necesarios, con los que completar esta acción. El contenido de este fichero, denominado *volumen_nfs_www.yaml*, es el siguiente:

```
/docker/resources/volumen_nfs_www.yaml

apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfswww
spec:
  capacity:
    storage: 15Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Recycle
  nfs:
    path: /volume1/dockerwww
    server: 10.130.120.95
```

Como se puede observar, los parámetros definidos son los indicados en la Tabla 3-4.

Definición del Volumen Servidor Web				
Tipo	Tamaño	Modo	Modo Persistencia	Tipo
PersistentVolume	15 GB	Lectura y escritura todos los nodos	Recycle	NFS

Tabla 3-4 Recursos definidos en el volumen “nfs – Servidor Web”. Autoría propia.

Creamos el volumen ejecutando el siguiente comando:

```
root$putin:~# kubectl create -f /docker/resources/volumen_nfs_www.yaml
```

Una vez se ha completado la operación, podemos visualizar su estado tecleando:

```
root$putin:~# kubectl get pv
```

Cuya salida se muestra a continuación en la Figura 3-11.

```
root@putin:~/resources# kubectl create -f volumen_nfs_www.yml
```

```
PersistentVolume/nfswww created
```

```
root@putin:~/resources# kubectl get pv
```

NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS	CLAIM	STORAGECLASS	AGE
nfswww	15Gi	RWX	Recycle	Available			9s

Figura 3-11. Estado creación volumen nfswww en el cluster Kubernetes. Autoría propia.

Continuando con la configuración de volúmenes, a continuación, crearemos otro de tipo *PersistentVolumeClaim*. Generamos el fichero de configuración denominado *volumen_nfs_www_pvc.yaml*

```
/docker/resources/volumen_nfs_pvc.yaml

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfswww-pvc
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 15Gi
```

Los parámetros definidos son los indicados en la Tabla 3-5.

Tipo	Tamaño	Modo
PersistentVolumeClaim	15 GB	Lectura y Escritura todos los nodos

Tabla 3-5 Recursos definidos en el volumen “nfswww-pvc”. Autoría propia.

Creamos el volumen ejecutando el siguiente comando:

```
root$putin:~# kubectl create -f /docker/resources/volumen_nfs_www_pvc.yaml
```

Para el contenedor que albergará el sistema de gestión de base de datos MariaDB, también es necesaria la creación de un volumen persistente. Configurado con los parámetros indicados en la Tabla 3-6.

Definición del Volumen Base de Datos				
Tipo	Tamaño	Modo	Modo Persistencia	Tipo
PersistentVolume	30 GB	Lectura y escritura todos los nodos	Recycle	NFS

Tabla 3-6 Recursos definidos en el volumen “nfs- servidor Base de Datos”. Autoría propia.

La configuración de este nuevo contenedor es la siguiente:

```
/docker/resources/volumen_nfs_bbdd.yaml

apiVersion: v1
kind: PersistentVolume
metadata:
```

```
name: nfsbdd
spec:
  capacity:
    storage: 30Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Recycle
  nfs:
    path: /volume1/dockerbdd
    server: 10.130.120.95
```

Creamos el volumen ejecutando el siguiente comando:

```
root$putin:~# kubectl create -f /docker/resources/volumen_nfs_bbdd.yaml
```

Por último, definimos el contenedor *PersistentVolumeClaim* asociado a éste (véase Tabla 3-7)

Tipo	Tamaño	Modo
PersistentVolumeClaim	30 GB	Lectura y Escritura todos los nodos

Tabla 3-7 Recursos definidos en el volumen “nfsbdd-pvc”. Autoría propia.

Creamos el volumen ejecutando el siguiente comando:

```
root$putin:~# kubectl create -f /docker/resources/volumen_nfs_bbdd_pvc.yaml
```

Visualizamos el estado de los volúmenes creados, obteniendo la salida mostrada en la Figura 3-12:

```
root$putin:~# kubectl get pv, pvc -o wide
```

```
root$putin:~# kubectl get pv,pvc -o wide
NAME                                CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS  CLAIM                                STORAGECLASS  REASON  AGE  VOLUMEMODE
persistentvolume/nfsbdd             15Gi      RWX           Recycle         Bound   default/nfsbdd-pvc                  4d          Filesystem
persistentvolume/nfswww             15Gi      RWX           Recycle         Bound   default/nfswww-pvc                  4d          Filesystem

NAME                                STATUS  VOLUME  CAPACITY  ACCESS MODES  STORAGECLASS  AGE  VOLUMEMODE
persistentvolumeclaim/nfsbdd-pvc     Bound   nfsbdd  15Gi      RWX           4d          Filesystem
persistentvolumeclaim/nfswww-pvc     Bound   nfswww  15Gi      RWX           4d          Filesystem
root$putin:~#
```

Figura 3-12. Volúmenes persistentes creados en el cluster Kubernetes. Autoría propia.

Una vez llegamos a este punto, tenemos desplegado y configurado un *cluster* Kubernetes habilitado para albergar los *pods* y *deployments* que ejecutarán, procesarán y almacenarán los datos de la aplicación.

3.3 Diseño de la aplicación

3.3.1 Requisitos y ámbito de aplicación

Como ya se comentó al principio de este documento, el segundo objetivo del presente trabajo es el diseño de una aplicación para la gestión integral de una flota de vehículos.

Esta aplicación se ha diseñado en un modelo de tres capas (MVC), utilizando el lenguaje de programación PHP, el servidor web Nginx y base de datos MariaDB (LEMP).

Los principales objetivos que se pretenden alcanzar con este sistema son los siguientes:

- Permitir administrar y registrar la flota de vehículos asignados a la Unidad.
- Poder realizar altas, bajas y actualizaciones de los vehículos adjudicados.
- Categorizar los vehículos en función de sus características.
- Facilitar la autonomía del Departamento de Apoyo Logístico, proporcionando una herramienta informática para la gestión de la flota de vehículos.
- Realizar consultas con los datos almacenados en el sistema.
- Posibilitar la extracción de listados sobre varios aspectos del estado de la flota de vehículos.
- Llevar a cabo la trazabilidad de los vehículos desde su adjudicación hasta su baja definitiva, registrando todos los acontecimientos, reparaciones, ITV, sanciones, siniestros, accidentes o robos que puedan sobrevenirse.
- Controlar y gestionar los repostajes efectuados por la flota de vehículos.
- Almacenar los datos básicos de los talleres y Organismos oficiales con los que se relaciona el Equipo de Apoyo Logístico para la gestión de la flota de vehículos
- Adjudicar, de forma automática, los vehículos recepcionados a los distintos Departamentos que componen la Unidad, en función de unos parámetros establecidos al efecto.

Como se observa en la Figura 3-13, el sistema administra información relativa a todos los elementos descritos en la misma.

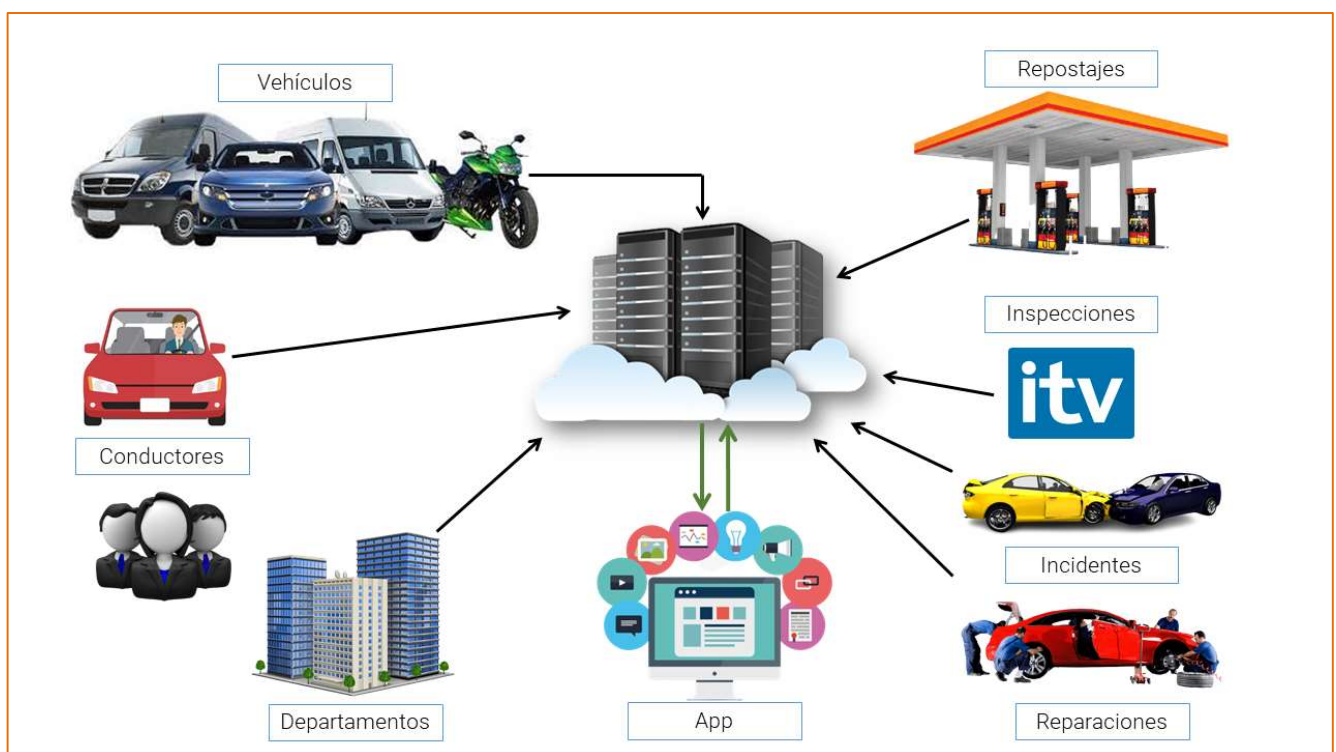


Figura 3-13. Entidades cuya información procesa el sistema. Autoría propia.

3.3.2 Establecimiento de requisitos

A continuación, como se muestra en la Tabla 3-8, se especifican los requisitos **funcionales** que debería contemplar la aplicación:

ID Requisito	Descripción Requisito
RF-VEH-001	El sistema deberá permitir dar de alta los vehículos adjudicados a la Unidad.
RF-VEH-002	Permitirá la modificación de cualquier dato de los vehículos registrados.

RF-VEH-003	Una vez que un vehículo no se encuentre operativo, será dado de baja en la aplicación.
RF-VEH-004	El sistema permitirá realizar consultas sobre un vehículo mediante sus datos básicos de registro.
RF-VEH-005	El sistema podrá vincular un vehículo con un Departamento y con una o varias personas pertenecientes a la Unidad.
RF-VEH-006	La aplicación visualizará la interrelación existente entre los vehículos, Departamentos y personal destinado en la Unidad.
RF-VEH-007	El sistema deberá gestionar todas las incidencias asociadas a cada vehículo.
RF-VEH-008	El sistema establecerá una utilidad para la generación de listados.
RF-VEH-009	El sistema proporcionará las interfaces necesarias para que los usuarios del Departamento de Apoyo Logístico vinculen los vehículos con el personal destinado en la Unidad.
RF-VEH-010	El sistema registrará los repostajes que efectúe un vehículo.
RF-VEH-011	El sistema estará equipado con un agente programado que pueda generar alertas sobre distintas eventualidades que puedan sobrevenir a los vehículos de la flota.
RF-VEH-012	Podrán definirse umbrales de alertas, sobre determinados aspectos relacionados con los vehículos y sus accesorios, de forma que los usuarios del Departamento de Apoyo Logístico puedan estar informados de fechas de caducidad para realizar cambios, reparaciones o inspecciones antes de la fecha de prescripción.
RF-VEH-013	El sistema dispondrá de un cuadro de mando en el que visualizar, de forma generalizada, el estado de la flota de vehículos.
RF-VEH-014	El sistema realizará la adjudicación automática de un nuevo vehículo.
RF-VEH-015	El sistema registrará datos sobre personas y Departamentos relacionados con un vehículo.
RF-VEH-016	El sistema realizará la autenticación y autorización de la entrada de usuarios a la aplicación.
RF-VEH-017	El sistema auditará, para todos los usuarios, las transacciones que realicen en el sistema.

Tabla 3-8 Recursos funcionales de la aplicación. Autoría propia.

En cuanto a los requisitos **no funcionales**, reflejados en la Tabla 3-9, la aplicación contemplará los siguientes:

ID Requisito	Descripción Requisito
RNF-VEH-001	El sistema se integrará en la política de <i>backups</i> del Servicio de Informática.
RNF-VEH-002	Los usuarios se autenticarán mediante los datos de su TIP (Tarjeta de Identidad Profesional)
RNF-VEH-003	El sistema utilizará, como gestor de base de datos, MariaDB.
RNF-VEH-004	El lenguaje de programación será PHP.
RNF-VEH-005	El sistema será diseñado para que cumpla con la Ley Orgánica de Protección de Datos.
RNF-VEH-006	La aplicación será capaz de funcionar en cualquier equipo informático homologado en la Unidad.

Tabla 3-9 Recursos no funcionales de la aplicación. Autoría propia.

3.3.1 Actores y especificaciones de los casos de uso

En cuanto a los actores que intervendrán en el sistema, se contemplan los siguientes:

- **Administrador:** usuario del sistema con privilegios elevados para llevar a cabo cualquier función del sistema y, en particular, la creación de usuarios, la adjudicación de vehículos y el control de auditoría.
- **Usuarios Avanzados:** integrantes del Equipo de Apoyo Logístico, pueden realizar cualquier tarea relacionada con la flota de vehículos, excepto la adjudicación de vehículos y creación de usuarios en el sistema.
- **Usuarios Departamentales:** tienen asignado un perfil básico, con el que pueden registrar repostajes, visualizar ficha de datos e historial de los vehículos asignados a su Departamento y enviar mensajes, para la realización de tareas de mantenimiento a los usuarios del Equipo de Apoyo Logístico, de los vehículos encuadrados en su Unidad.
- **Auditor:** sistema establecido para el mantenimiento de un registro de acceso y actividades que realizan los usuarios en el sistema.

Listados de casos de uso:

- Relacionados con la administración del sistema
 - Gestionar usuarios
 - Comprobar auditoría
 - Adjudicación de vehículos
- Relacionados con los usuarios avanzados:
 - Gestión de vehículos (altas, bajas, modificaciones, etc.)
 - Vinculación de vehículos con Departamentos y conductores.
 - Tramitación de inspecciones, reparaciones e incidentes.
 - Administración y control del repostaje.
 - Gestión de accesorios y elementos auxiliares.
- Relacionados con los usuarios departamentales:
 - Registro de los repostajes realizados.
 - Visualización ficha técnica de los vehículos asignados.
 - Petición de tareas de mantenimiento y envío de mensajes al Equipo de Apoyo Logístico.

A continuación, en la Figura 3-14, se muestra el diagrama de casos de uso global:

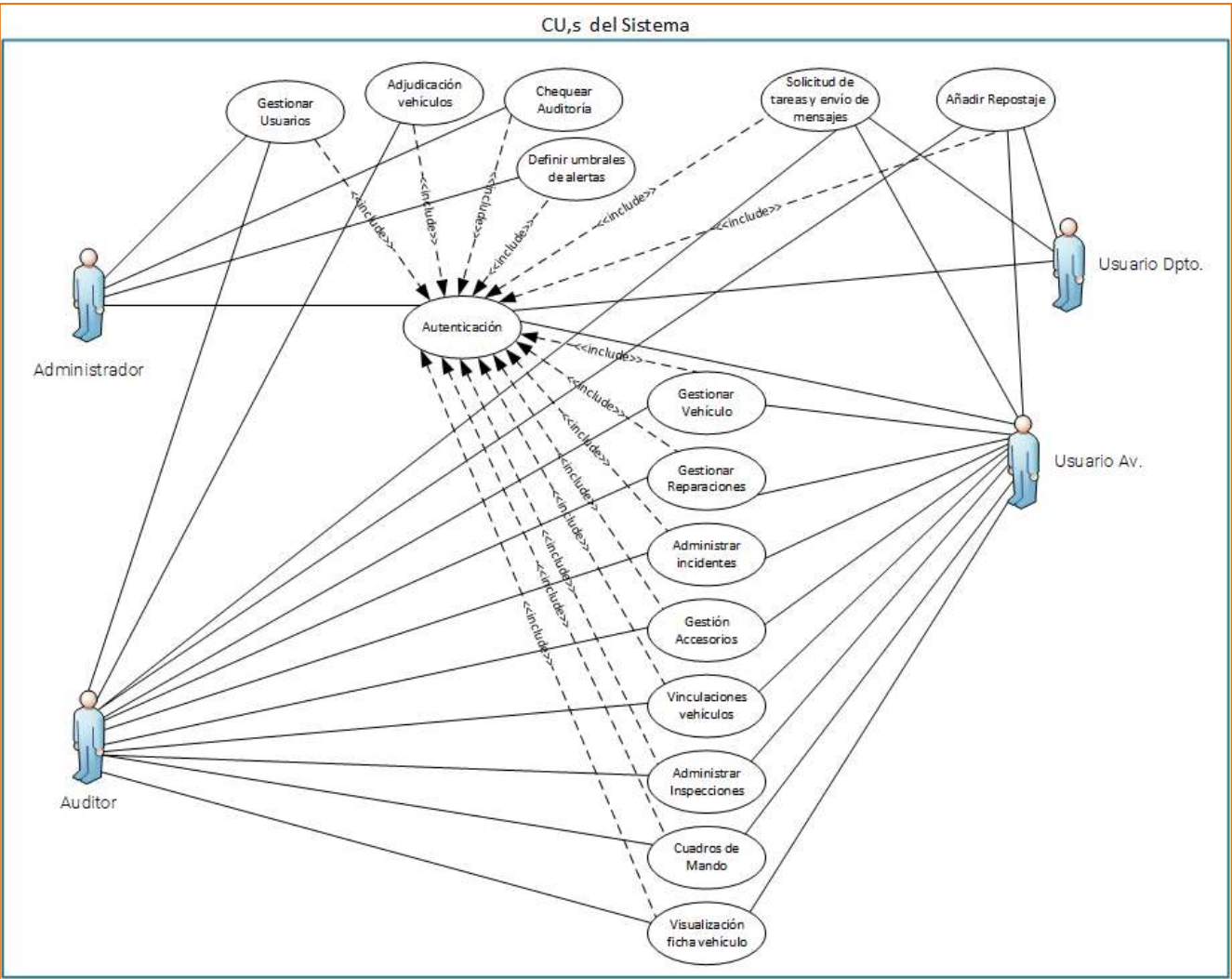


Figura 3-14. Casos de Uso global del sistema. Autoría propia.

3.3.2 Descripción de los casos de uso

Si bien, la finalidad de este trabajo es la definición y despliegue de una aplicación para la gestión de una flota de vehículos, por no extendernos en el diseño de todos los módulos, componentes, casos de uso, etc., solamente se describen, de forma general, algunos de los casos de uso del sistema reflejados en la Figura 3-14, para los usuarios “avanzado”, “administrador” y “departamental”.

A continuación, en la Tabla 3-10, se detallan los casos de uso y la interacción del *usuario avanzado* y del *usuario departamental* con el sistema.

Flujo de Eventos: CU-01: Autenticación	
Descripción: El usuario avanzado desea conectarse al sistema y debe proporcionar sus datos de entrada (nombre de usuario y contraseña)	
Actores: Usuario avanzado, usuario departamental y auditor.	
ACTOR	SISTEMA
1. El usuario accede a la aplicación con su navegador y se muestra la pantalla de inicio.	2. El sistema presenta la pantalla de “Login” y le solicita el nombre de usuario y la contraseña.

3. El usuario teclea sus datos y pulsa en la opción “Entrar”	4. Si los datos introducidos son correctos, el sistema muestra la pantalla con el menú principal de opciones para el usuario avanzado.
5. Se invoca el sistema de auditoría para registrar la acción del usuario.	6. El sistema auditor almacena, en la base de datos, un registro con indicación del módulo “Entrada al sistema”, el nombre del usuario, la fecha y hora y el texto “Entrada correcta al sistema”.

Flujo de Eventos: CU-02: Gestionar vehículos

Descripción: El usuario avanzado selecciona la opción “Gestionar Vehículo” en la pantalla principal de su menú, pudiendo crear, eliminar, consultar o modificar los datos de los vehículos de la flota.

Actores: Usuario avanzado y auditor.

ACTOR	SISTEMA
1. El usuario avanzado pulsa el botón “Gestionar vehículos” en su pantalla de inicio.	2. El sistema presenta la pantalla de Gestión correspondiente con las distintas opciones (alta, baja, modificación y sincronización de vehículos).
3. El usuario selecciona la opción “Alta” de un vehículo	4. El sistema muestra la pantalla correspondiente para realizar el alta.
5. El usuario avanzado completa los campos y pulsa “Guardar”.	6. El sistema muestra el mensaje de “Vehículo almacenado correctamente”, y vuelve a la pantalla inicial de Gestión de vehículos.
7. Se invoca el sistema de auditoría para registrar la acción del usuario.	8. El sistema auditor almacena, en la base de datos, un registro con indicación del módulo “Gestionar vehículo”, el nombre del usuario, la fecha y hora y el tipo de acción realizada (alta, baja, modificación o consulta).

Flujo de Eventos: CU-03: Gestionar reparaciones

Descripción: El usuario avanzado selecciona la opción “Gestionar Vehículo” en la pantalla principal de su menú, pudiendo crear, eliminar o modificar los datos de una reparación que se ha solicitado de un vehículo.

Actores: Usuario avanzado y auditor.

ACTOR	SISTEMA
1. El usuario avanzado pulsa el botón “Gestionar Reparaciones” en su pantalla de inicio.	2. El sistema presenta la pantalla de Gestión correspondiente con las distintas opciones (alta, baja, modificación y consulta). En la misma pantalla se muestran opciones de “Gestión de talleres”, desde donde se pueden añadir, modificar o consultar los talleres con los que trabaja el Grupo de Apoyo Logístico.
3. El usuario selecciona la opción “Alta” de una reparación.	4. El sistema muestra la pantalla correspondiente para realizar el alta.

5. El usuario avanzado completa la información de los distintos apartados y pulsa “Guardar”.	6. El sistema muestra el mensaje de “Reparación almacenada correctamente”, y vuelve a la pantalla inicial de Gestión de Reparaciones.
7. Se invoca el sistema de auditoría para registrar la acción del usuario.	8. El sistema auditor almacena, en la base de datos, un registro con indicación del módulo “Gestionar reparaciones”, el nombre del usuario, la fecha y hora y el tipo de acción realizada (alta, baja, modificación o consulta).
Flujo de Eventos: CU-04: Administrar Incidentes	
Descripción: El usuario avanzado selecciona la opción “Administrar incidentes” en la pantalla principal de su menú, pudiendo crear, modificar o consultar los datos un incidente en el que se ha visto involucrado un vehículo de la flota.	
Actores: Usuario avanzado y auditor.	
ACTOR	SISTEMA
1. El usuario avanzado pulsa el botón de “Administrar incidentes” en su pantalla de inicio.	2. El sistema presenta la pantalla de Gestión correspondiente con las distintas opciones (alta, baja, modificación y consulta).
3. El usuario selecciona la opción “Alta” de un incidente.	4. El sistema muestra la pantalla correspondiente para realizar el alta.
5. El usuario avanzado completa los apartados y pulsa “Guardar”.	6. El sistema muestra el mensaje de “Incidente registrado correctamente”, y vuelve a la pantalla inicial de Gestión de incidentes.
7. Se invoca el sistema de auditoría para registrar la acción del usuario.	8. El sistema auditor almacena, en la base de datos, un registro con indicación del módulo “Administrar incidentes”, el nombre del usuario, la fecha y hora y el tipo de acción realizada (alta, baja, modificación o consulta).
Flujo de Eventos: CU-05: Gestión de Accesorios	
Descripción: El usuario avanzado selecciona la opción de “Gestión de accesorios” en la pantalla principal de su menú, pudiendo crear, modificar, consultar o dar de baja los accesorios que pueden vincularse con un vehículo de la flota.	
Actores: Usuario avanzado y auditor.	
ACTOR	SISTEMA
1. El usuario avanzado pulsa el botón de “Gestión de accesorios” en su pantalla de inicio.	2. El sistema presenta la pantalla de Gestión correspondiente con las distintas opciones (alta, baja, modificación y consulta).
3. El usuario selecciona la opción “Alta” de un accesorio.	4. El sistema muestra la pantalla correspondiente para realizar el alta.
5. El usuario avanzado rellena los campos y pulsa “Guardar”.	6. El sistema muestra el mensaje de “Accesorio registrado correctamente”, y vuelve a la pantalla inicial de Gestión de accesorios.

7. Se invoca el sistema de auditoría para registrar la acción del usuario.	8. El sistema auditor almacena, en la base de datos, un registro con indicación del módulo “Gestión de accesorios”, el nombre del usuario, la fecha y hora y el tipo de acción realizada (alta, baja, modificación o consulta).
--	---

Flujo de Eventos: CU-06: Vinculaciones Vehículos

Descripción: El usuario avanzado selecciona la opción de “Vinculaciones vehículos” en la pantalla principal de su menú, pudiendo crear, modificar, consultar o dar de baja las distintas vinculaciones de los distintos vehículos de la flota con Unidades, conductores, reparaciones, incidentes, accesorios e inspecciones.

Actores: Usuario avanzado y auditor.

ACTOR	SISTEMA
1. El usuario avanzado pulsa el botón de “Vinculaciones vehículos” en su pantalla de inicio.	2. El sistema presenta la pantalla correspondiente, con un listado de todos los vehículos registrados, y las opciones “Vincular” o “Desvincular”.
3. El usuario avanzado selecciona un vehículo del listado y, a continuación, pulsa en la opción “Vincular”.	4. El sistema muestra una pantalla con los objetos vinculados actualmente al vehículo y un apartado con los distintos elementos que pueden vincularse.
5. El usuario avanzado selecciona un objeto (Unidad, conductor, accesorio, reparación, incidente o inspección) y pulsa “Continuar”.	6. El sistema muestra los distintos elementos que forman parte del objeto seleccionado. En el caso de Unidades, muestra un listado con todos los Departamentos y Grupos que componen la Unidad.
7. El usuario Administrador selecciona un Departamento o Grupo y pulsa en Vincular.	8. El sistema muestra el mensaje “Unidad vinculada correctamente con el vehículo” y vuelve a la pantalla de Vinculaciones vehículos.
9. Se invoca el sistema de auditoría para registrar la acción del usuario.	10. El sistema auditor almacena, en la base de datos, un registro con indicación del módulo “Vincular vehículos”, el nombre del usuario, la fecha y hora y el tipo de acción realizada (vincular o desvincular y el objeto utilizado).

Flujo de Eventos: CU-07: Administrar inspecciones

Descripción: El usuario avanzado selecciona la opción de “Administrar inspecciones” en la pantalla principal de su menú, pudiendo crear, modificar, consultar o dar de baja las inspecciones mecánicas e ITV que pueden realizarse con un vehículo de la flota.

Actores: Usuario Avanzado y Auditor.

ACTOR	SISTEMA
1. El usuario avanzado pulsa el botón de “Administrar inspecciones” en su pantalla de inicio.	2. El sistema presenta la pantalla de Gestión correspondiente con las distintas opciones (alta, baja, modificación y consulta).
3. El usuario selecciona la opción “Alta” de una Inspección.	4. El sistema muestra la pantalla correspondiente para realizar el alta.

5. El usuario avanzado completa la información de los distintos campos y pulsa “Guardar”.	6. El sistema muestra el mensaje de “Inspección guardada correctamente”, y vuelve a la pantalla inicial de Administrar inspecciones.
7. Se invoca el sistema de auditoría para registrar la acción del usuario.	8. El sistema auditor almacena, en la base de datos, un registro con indicación del módulo “Administrar inspecciones”, el nombre del usuario, la fecha y hora y el tipo de acción realizada (alta, baja, modificación o consulta).
Flujo de Eventos: CU-08: Cuadros de Mando	
Descripción: El usuario avanzado selecciona la opción de “Cuadros de mando” en la pantalla principal de su menú, pudiendo visualizar un panel con información estadística relativa a los vehículos de la flota, como pueden ser: Estado actual de los vehículos, kilómetros recorridos en el mes actual, número de vehículos por tipo, número de vehículos asignados a los distintos Departamentos y Grupos, etc.	
Actores: Usuario avanzado y auditor.	
ACTOR	SISTEMA
1. El usuario avanzado pulsa el botón de “Cuadros de mando” en su pantalla de inicio.	2. El sistema presenta la pantalla con varios cuadros de mando “pre-diseñados”, pudiendo seleccionar uno de ellos.
3. El usuario selecciona el cuadro de mando “Principal”.	4. El sistema muestra la pantalla correspondiente al cuadro de mando Principal mostrando información de: - Número de vehículos por tipo de vehículo. - Kilómetros recorridos en el mes actual. - Estado general de los vehículos. - Número de reparaciones, incidentes e inspecciones llevados a cabo en el mes actual.
Flujo de Eventos: CU-09: Visualización ficha vehículo	
Descripción: El usuario avanzado selecciona la opción de “Visualización ficha vehículo” en la pantalla principal de su menú. Selecciona un vehículo del listado y accede a toda la información relativa al vehículo, como puede ser Departamento y conductor asignado, reparaciones efectuadas, inspecciones llevadas a cabo, incidentes, repostajes, etc.	
Actores: Usuario avanzado y auditor.	
ACTOR	SISTEMA
1. El usuario avanzado pulsa el botón de “Visualización ficha vehículo” en su pantalla de inicio.	2. El sistema presenta en pantalla un listado de todos los vehículos de la Unidad.
3. El usuario selecciona un vehículo del listado y pulsa en “Visualizar”.	4. El sistema muestra la pantalla correspondiente del vehículo con información relativa a: - Datos básicos del vehículo. - Histórico de reparaciones efectuadas.

	• Histórico de incidentes. • Accesorios vinculados al vehículo. • Histórico de inspecciones. • Repostajes realizados en los últimos dos meses.
5. Se invoca el sistema de auditoría para registrar la acción del usuario.	6. El sistema auditor almacena, en la base de datos, un registro con indicación del módulo “Visualizar ficha vehículo”, el nombre del usuario, la fecha y hora y el identificador del vehículo consultado.

Flujo de Eventos: CU-10: Añadir Repostaje

Descripción: El usuario avanzado selecciona la opción “Añadir repostaje” en la pantalla principal de su menú, mediante la cual actualiza datos relativos al kilometraje actual que tiene el vehículo en cuestión y número de repostajes efectuados, entre otros datos.

Actores: Usuario avanzado, usuario departamental y auditor.

ACTOR	SISTEMA
1. El usuario pulsa el botón “Añadir repostaje” en su pantalla de inicio.	2. El sistema presenta la pantalla de Repostajes con las distintas opciones (nuevo, modificar, borrar y consultar).
3. El usuario selecciona la opción “Nuevo repostaje”.	4. El sistema muestra la pantalla correspondiente en la que solicita los siguientes datos al usuario: 1. Número ID del vehículo. 2. Kilómetros actuales en el momento del repostaje. 3. Tipo de combustible. 4. Cantidad en litros. 5. Importe total. 6. Tipo de pago (tarjeta, efectivo o cheque). 7. Estación de servicio y dirección. 8. Coordenadas GPS.
5. El usuario completa la información solicitada y pulsa “Guardar”.	6. El sistema muestra el mensaje de “Repostaje registrado correctamente”, y vuelve a la pantalla inicial de Añadir repostaje.
7. Se invoca el sistema de auditoría para registrar la acción del usuario.	8. El sistema auditor almacena, en la base de datos, un registro con indicación del módulo “Añadir Repostaje”, el nombre del usuario, la fecha y hora y la acción realizada.

Flujo de Eventos: CU-11: Solicitud de tareas y envío de mensajes

Descripción: El usuario avanzado selecciona la opción “Solicitud de tareas y envío de mensajes” en la pantalla principal de su menú. El sistema le permite, bien registrar una solicitud de trabajo al Equipo de Apoyo Logístico (reparación, asistencia, envío de una grúa) o bien la emisión de un mensaje a un miembro del Equipo de Apoyo Logístico o al grupo al completo.

Actores: Usuario avanzado, usuario departamental y auditor.

ACTOR	SISTEMA
-------	---------

1. El usuario pulsa el botón “Solicitud de tareas y envío de mensajes” en su pantalla de inicio.	2. El sistema presenta una pantalla, dividida en dos apartados verticales, en cada uno de los cuales se visualiza la posibilidad de solicitar una tarea de apoyo (reparación, asistencia o envío de una grúa) y en la siguiente la posibilidad de enviar un mensaje a un miembro del Grupo de Apoyo Logístico o bien a todo el grupo.
3. El usuario selecciona la opción y completa los apartados para enviar un mensaje a todo el Grupo de Apoyo Logístico. Completa el texto del mensaje y pulsa en “Enviar”.	4. El sistema pregunta al usuario si los datos introducidos son correctos. Si pulsa en “Sí”, el sistema enviará el mensaje tecleado a todos al usuario general del Grupo de Apoyo Logístico. Una vez enviado, el sistema regresa al menú principal.
5. Se invoca el sistema de auditoría para registrar la acción del usuario.	6. El sistema auditor almacena, en la base de datos, un registro con indicación del módulo “Envío de Mensaje”, el nombre del usuario, la fecha y hora y el texto del mensaje enviado.

Tabla 3-10 Definición casos de uso del usuario avanzado. Autoría propia.

Seguidamente, en la Tabla 3-11, se detallan los casos de uso para el usuario *administrador*.

Flujo de Eventos: CU-12: Autenticación	
Descripción: El usuario administrador desea conectarse al sistema y debe proporcionar sus datos de entrada (nombre de usuario y contraseña)	
Actores: Usuario administrador y auditor.	
ACTOR	SISTEMA
1. El usuario administrador accede a la aplicación con su navegador y se muestra la pantalla de inicio.	2. El sistema presenta la pantalla de “Login” y le solicita el nombre de usuario y la contraseña.
3. El usuario teclea sus datos y pulsa en la opción “Entrar”	4. Si los datos introducidos son correctos, el sistema muestra la pantalla con el menú principal de opciones para el usuario administrador.
5. Se invoca el sistema de auditoría para registrar la acción del usuario.	6. El sistema auditor almacena, en la base de datos, un registro con indicación del módulo “Entrada al sistema”, el nombre del usuario, la fecha y hora y el texto “Entrada correcta al sistema”.
Flujo de Eventos: CU-13: Gestionar usuarios	
Descripción: El usuario administrador selecciona la opción “Gestionar usuarios” en la pantalla principal de su menú, pudiendo dar de alta, eliminar, consultar o modificar los datos de los usuarios de la aplicación.	
Actores: Usuario administrador y auditor.	
ACTOR	SISTEMA

1. El usuario administrador pulsa el botón “Gestionar usuarios” en su pantalla de inicio.	2. El sistema presenta la pantalla de gestión correspondiente con las distintas opciones (alta, baja, modificación y consulta de usuarios).
3. El usuario selecciona la opción “Alta” de un usuario	4. El sistema muestra la pantalla correspondiente para realizar el alta.
5. El usuario administrador completa la información de los campos y pulsa “Guardar”.	6. El sistema muestra el mensaje de “Usuario almacenado correctamente”, y vuelve a la pantalla inicial de gestión de usuarios.
7. Se invoca el sistema de auditoría para registrar la acción del usuario.	8. El sistema auditor almacena, en la base de datos, un registro con indicación del módulo “Gestionar usuarios”, el nombre del usuario administrador, la fecha y hora y el tipo de acción realizada (alta, baja, modificación o consulta).

Flujo de Eventos: CU-14: Adjudicación vehículos

Descripción: El usuario administrador selecciona la opción “Adjudicación vehículos” en la pantalla principal de su menú, lo que activa una serie de procesos con los que la aplicación clasifica los vehículos en función de sus características, recopila información sobre los vehículos asignados a los diferentes Departamentos y Grupos, consulta la información del personal integrante de cada uno de ellos y, finalmente, adjudica el vehículo al Grupo que menor ratio de adjudicaciones posee en ese momento.

Actores: Usuario administrador y auditor.

ACTOR	SISTEMA
1. El usuario administrador pulsa el botón de “Adjudicación vehículos” en el menú de opciones y en la pantalla de adjudicación de vehículos.	2. El sistema ejecuta los siguientes procesos de forma encadenada: 1. Busca los vehículos que todavía no se encuentran adjudicados y han sido dados de alta en el sistema. Estos vehículos han sido previamente clasificados en las distintas categorías definidas en el sistema. 2. Realiza una consulta sobre el número de integrantes de cada Departamento y Grupo (exceptuando puestos vacantes), y calcula la ratio de puntuación en relación a los puestos de trabajo definidos en la plantilla de la Unidad. 3. Selecciona el primer vehículo de la lista para su asignación, consulta su “categoría y clase” y lo asigna al Grupo con menor ratio de ocupación de la Unidad. 4. Continúa con el resto de vehículos hasta finalizar el reparto. Finalmente, muestra en pantalla un listado con los vehículos que pueden ser adjudicados y los grupos a los que serían entregados.

3. El usuario administrador, si observa que todo se ha realizado correctamente, pulsará en la opción “Adjudicar”. En caso contrario, pulsará en la opción “Anular adjudicación”, regresando a la pantalla principal.	4. El sistema realiza las adjudicaciones definitivas de los vehículos del listado y muestra un mensaje con el texto “Vehículos adjudicados correctamente”.
5. El usuario administrador pulsa en la opción “Notificar” en su pantalla de trabajo.	6. El sistema envía un correo a los usuarios definidos en cada uno de los grupos para informarles de las nuevas adjudicaciones que les han sido asignadas.
7. Se invoca el sistema de auditoría para registrar la acción del usuario.	8. El sistema auditor almacena, en la base de datos, un registro con indicación del módulo “Adjudicación de vehículos”, el nombre del usuario administrador, la fecha, hora, los números de los vehículos asignados y el código del Grupo al que ha sido asignado.

Flujo de Eventos: CU-15: Chequear auditoría

Descripción: El usuario administrador selecciona la opción “Comprobar auditoría” en la pantalla principal de su menú, con la que accede al registro de “logs” de las actividades del sistema.

Actores: Usuario administrador y auditor.

ACTOR	SISTEMA
1. El usuario administrador pulsa el botón “Comprobar auditoría” en su pantalla de inicio.	2. El sistema presenta una pantalla en la que puede seleccionarse la fecha de inicio, fecha final, nombre de usuario, módulo del sistema y el tipo de acción del sistema.
3. El usuario completa los apartados necesarios y pulsa “Buscar”.	4. El sistema muestra un listado con las entradas que se corresponden con los parámetros designados por el usuario administrador.
5. Se invoca el sistema de auditoría para registrar la acción del usuario.	6. El sistema auditor almacena, en la base de datos, un registro con indicación del módulo “Comprobar auditoría”, el nombre del usuario administrador, la fecha y hora y el texto “Consulta de auditoría del sistema”.

Flujo de Eventos: CU-16: Definir umbrales de alertas

Descripción: El usuario administrador selecciona la opción “Definir umbrales de alertas” en la pantalla principal de su menú, con la que accede al subsistema de definición de alertas. Este módulo almacena unos valores límite que, una vez alcanzados por los vehículos de la flota, difunden alarmas a los usuarios avanzados para su conocimiento y puesta en marcha de los trabajos y labores tendentes a su normalización.

Actores: Usuario administrador y auditor.

ACTOR	SISTEMA
1. El usuario administrador pulsa el botón “Definir umbrales de alertas” en su pantalla de inicio.	2. El sistema presenta la pantalla de gestión correspondiente con las distintas opciones (alta, baja, modificación y consulta de alertas).

3. El usuario selecciona la opción “Alta” de una alerta.	4. El sistema muestra la pantalla correspondiente para realizar el alta.
5. El usuario administrador completa la información de los campos y pulsa “Guardar”.	6. El sistema muestra el mensaje de “Alerta almacenada correctamente”, y vuelve a la pantalla inicial de definición de umbrales de alerta.
7. Se invoca el sistema de auditoría para registrar la acción del usuario.	8. El sistema auditor almacena, en la base de datos, un registro con indicación del módulo “Definir umbrales de alerta”, el nombre del usuario administrador, la fecha y hora y el tipo de acción realizado (alta, baja, modificación o consulta).

Tabla 3-11 Definición casos de uso del usuario administrador. Autoría propia.

3.3.3 Procedimiento Adjudicación de vehículos.

De todos los casos de uso que se han especificado en esta aplicación, el que presenta una mayor complejidad y cuyo resultado conlleva la ejecución de varios procesos concatenados, es el definido como “Adjudicación de vehículos”. Como se expuso en la descripción de este trabajo, el procedimiento de adjudicación debe ser transparente para todos los Departamentos que forman parte de la Organización, imparcial en su labor de establecimiento de cupos y con la posibilidad de ser auditado.

En la adjudicación de un vehículo intervienen diversos factores, como son:

- Número de integrantes de un Departamento o Grupo (puestos de trabajo)
- Puestos de trabajo que se encuentran asignados y con personal designado, en el momento de la adjudicación. No se tiene en cuenta el catálogo total, sino el número de integrantes que están activos en el momento del reparto.
- Roles del equipo de trabajo:
 - Jefe de Departamento o Grupo.
 - Usuario de investigación.
 - Usuario de elaboración y análisis.
 - Usuario administrativo.
- Puntuación asignada a cada uno de los “roles” definidos.

Los puestos de trabajo, definidos en la estructura de la Unidad, tienen una puntuación en relación a la función del puesto de trabajo. Un ejemplo es el que se muestra en la Tabla 3-12.

Puntuación puestos de trabajo				
Jefe_Área	Esp. Oper.	Obtención	Analista	Administrativo
1	0,6	0,5	0,3	0,2

Tabla 3-12 Puntuación puestos de trabajo. Autoría propia.

En cuanto a la plantilla, el ejemplo de una Organización podría ser como el que se muestra en la Tabla 3-13:

Dpto.	Grupo	Jefe_Área	Oper.	Obtención	Analista	Admin.	Actual	Vacantes	Total
A	A-1	1	5	7	3	2	18	5	23
A	A-2	1	10	3	3	1	18	1	19
B	B-1	2	15	10	4	1	32	4	36
B	B-2	1	2	10	2	2	17	1	18
B	B-3	2	0	20	2	3	27	3	30

Tabla 3-13 Ejemplo de Unidad, plantilla y catálogo de los puestos de trabajo. Autoría propia.

Finalmente, para realizar la clasificación de los vehículos, se tienen en cuenta variables como las contempladas en la Tabla 3-14:

Características	Clase A	Clase B	Clase C
Cilindrada	Mayor de 2000 cc	Entre 1400 y 2000 cc	Menor de 1400 cc
Potencia	Mayor de 200 CV	Entre 90 y 200 CV	Menor de 90 CV
Kilometraje	Sin limitación	Entre 10.000 y 300.000 kms	Mayor de 300.000 kms
Modelo	Todoterreno 4x4 Gama Alta Deportivo SUV Eléctrico	Berlina Monovolumen Pickup Crossover Coupé Híbrido	Sedan Micro Van Furgoneta Caravana Camión
Color	Blanco Gris Azul oscuro Negro	Rojo Amarillo Azul claro Naranja Violeta	Sin especificar
KW	Menor de 90 kW	Entre 90 y 130 kW	Mayor de 130 kW
Antigüedad Matrícula	Menor de 2 años	Entre 2 y 8 años	Mayor de 8 años

Tabla 3-14 Factores para la clasificación de los vehículos. Autoría propia.

Como se ha especificado en la definición del caso de uso *CU-14*, la adjudicación de un vehículo requiere la culminación de varios procesos secuenciales que comienzan con la búsqueda de los vehículos que se han introducido recientemente en la base de datos, y se encuentran sin asignar, finalizando con su adjudicación a los Departamentos y Grupos que menor ratio tienen, en relación a su plantilla y catalogación de sus puestos de trabajo.

En el caso que un vehículo obtenga diversas calificaciones, a la hora de ponderar los factores definidos en la Figura 3-14, el sistema utilizará una lista de condiciones establecidas por la Jefatura de la Unidad en cuanto a establecer su categoría final. Si un vehículo, por ejemplo, obtiene en su clasificación tres valoraciones de categoría “A” y tres valoraciones de categoría “B”, el vehículo será finalmente clasificado en la categoría “B”.

El procedimiento que debe llevarse a cabo, así como los procesos o piezas de código que intervienen en el mismo, son los que pueden observarse en la Figura 3-15.

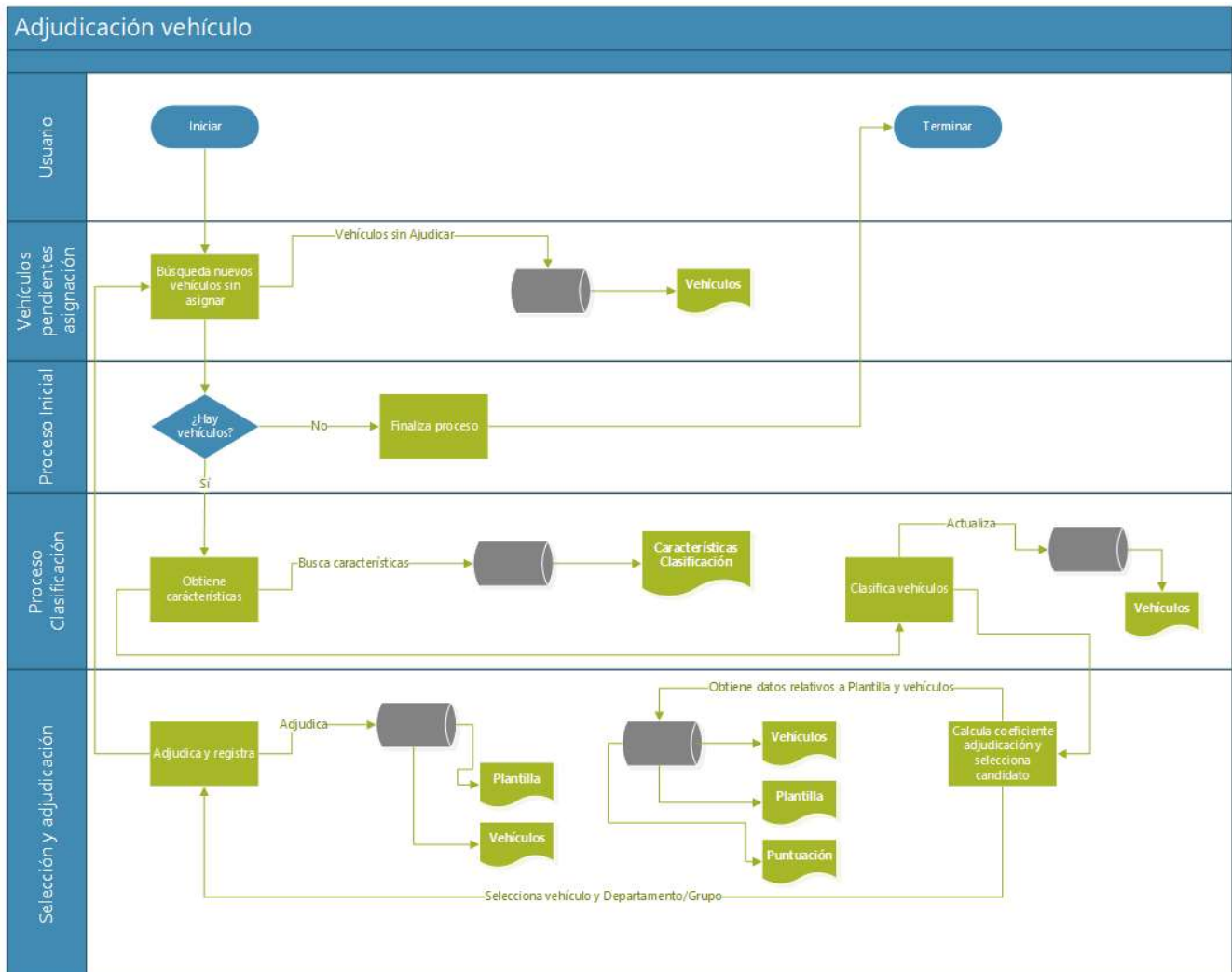


Figura 3-15. Procedimiento adjudicación nuevo vehículo. Autoría propia.

4 RESULTADOS / VALIDACIÓN

4.1 Despliegue de la plataforma

4.1.1 Despliegue servicio Web

Comenzamos el despliegue con los contenedores que albergarán el servidor web *nginx*. Todo el despliegue de contenedores lo llevaremos a cabo desde el servidor *putin*, quien tiene asignado el rol de *master* del *cluster*. Para ello, generamos un fichero *yaml* con la siguiente configuración:

```
/docker/resources/nginx-deployment-volume.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: nginx
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      creationTimestamp: null
    labels:
      app: nginx
    spec:
      containers:
      - image: nginx:1.14.2
```

```

imagePullPolicy: IfNotPresent
name: nginx
ports:
- containerPort: 80
  protocol: TCP
volumeMounts:
- name: nfswww
  mountPath: /usr/share/nginx/html
volumes:
- name: nfswww
  persistentVolumeClaim:
    claimName: nfswww-pvc

```

/docker/resources/nginx-pod.yaml

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-config
data:
  nginx.conf: |
    events {}
    http {
      server {
        listen 80;
        worker_processes 3;
        error_log /var/log/nginx/error_r.log;
        index index.php index.htm index.html;
        root /usr/share/nginx/html;
        location ~ \.php$ {
          include fastcgi_params;
          fastcgi_param REQUEST_METHOD $request_method;
          fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
          fastcgi_pass 127.0.0.1:9000;
        }
      }
    }
---
apiVersion: v1
kind: Pod
metadata:
  name: nginx-phpfpm
labels:
  app: nginx-phpfpm
  tier: back-end
spec:
  volumes:
  - name: nfswww
    persistentVolumeClaim:

```

```

    claimName: nfswww-pvc
  - name: nginx-config-volume
    configMap:
      name: nginx-config
containers:
  - name: nginx-container
    image: nginx:latest
    volumeMounts:
      - name: nginx-config-volume
        mountPath: /etc/nginx/nginx.conf
        subPath: nginx.conf
      - name: nfswww
        mountPath: /usr/share/nginx/html
  - name: php-fpm-container
    image: php:7.0-fpm

```

Como se puede observar, lo que estamos desplegando es un *pod* con el contenedor *nginx*, al que configuramos el puerto 80 como puerto de escucha local y, como almacenamiento, mapeamos su carpeta por defecto */usr/share/nginx/html* al volumen persistente definido como *nfswww-pvc*.

Desplegamos los contenedores ejecutando el siguiente comando:

```
root$putin:~# kubectl apply -f nginx-pod.yaml
```

Una vez ejecutado, podemos visualizar el resultado como se observa en la Figura 4-1.

```

root@putin:~/resources/mariadb# kubectl get pod -o wide
NAME          READY   STATUS    RESTARTS   AGE   IP           NODE   NOMINATED NODE   READINESS GATES
nginx-phpfpm  2/2     Running   2 (21h ago)  2d22h  10.244.2.14  merkel  <none>           <none>

```

Figura 4-1 Despliegue contenedores *nginx*. Autoría propia.

Posteriormente, para que la aplicación pueda ser accesible por los clientes ubicados fuera del *cluster*, definimos un servicio, que se encargará de conectar a los clientes con los contenedores que ejecutan el servidor web *nginx*. El tipo de servicio definido es el conocido como LoadBalancer, el cual permite interactuar con un balanceador externo y configurar una IP de servicio para acceder a la aplicación. En nuestro caso, y al tratarse de un laboratorio, configuramos la IP del servidor *putin* como si fuese la IP de servicio, de forma que desde el exterior pueda accederse por esta comunicación.

Para ello, definimos el siguiente fichero de configuración:

```

/docker/resources/nginx-service.yaml

apiVersion: v1
kind: Service
metadata:
  labels:
    app: nginx
  name: nginx-service
spec:
  ports:
    - name: 80-80
      nodePort: 30858
      port: 80

```

```

protocol: TCP
targetPort: 80
selector:
  app: nginx
type: LoadBalancer
externalIPs:
- 10.130.120.105

```

Y, aplicamos los cambios. Una vez completados, podemos observar el resultado consultando el estado de los servicios desplegados en el *cluster*, ejecutando los comandos:

```

root$putin:~# kubectl get svc -o wide
root$putin:~# kubectl get describe svc nginx-service

```

Obteniendo la información que se muestra en la Figura 4-2.

```

root@putin:~/resources# kubectl get svc -o wide
NAME                TYPE          CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE      SELECTOR
kubernetes           ClusterIP      10.96.0.1        <none>            443/TCP          15d      <none>
nginx-service        LoadBalancer  10.102.235.67    10.130.120.105   80:30858/TCP     2m20s    app=nginx
root@putin:~/resources# kubectl describe svc nginx-service
Name:                nginx-service
Namespace:           default
Labels:              app=nginx
Annotations:          <none>
Selector:             app=nginx
Type:                LoadBalancer
IP Family Policy:    SingleStack
IP Families:         IPv4
IP:                  10.102.235.67
IPs:                 10.102.235.67
External IPs:        10.130.120.105
Port:                80-80  80/TCP
TargetPort:          80/TCP
NodePort:            80-80  30858/TCP
Endpoints:           10.244.1.4:80,10.244.1.5:80,10.244.2.5:80
Session Affinity:    None
External Traffic Policy: Cluster
Events:              <none>

```

Figura 4-2 Despliegue contenedores nginx. Autoría propia.

4.1.2 Despliegue sistema gestor de base de datos

Para el despliegue del contenedor que albergará el gestor de base de datos MariaDB, necesitamos configurar un usuario y contraseña de acceso a la base de datos, previamente al despliegue del contenedor. Para ello generamos tanto el usuario, como la contraseña en *base64*, de forma que quede ofuscada la información en el fichero *yaml*. El contenido del fichero es el siguiente:

```

/docker/resources/mariadb-secret.yaml

apiVersion: v1
data:
  password: cGF0b3BhdG8=
  username: cm9vdA==
  mariadb-root-password: cGF0b3BhdG8=
kind: Secret
metadata:
  name: mariadb-secret
  namespace: default
type: Opaque

```

También definimos un recurso *ConfigMap* para almacenar pares de clave/valores no confidenciales y que puedan ser utilizados como variables de entorno, argumentos de línea de comandos o como archivos de configuración en un volumen. El fichero tiene la siguiente información:

/docker/resources/mariadb-configmap.yaml
<pre> apiVersion: v1 kind: ConfigMap metadata: name: mariadb-configmap data: database_url: mariadb-internal-service </pre>

Y, por último, el fichero de arranque de los contenedores con el sistema gestor de base de datos MariaDB, al que hemos añadido otro contenedor con el program phpMyAdmin [47], con el que poder acceder a la administración y diseño de la base de datos de la aplicación. Este fichero es el siguiente:

/docker/resources/mariadb-pod.yaml
<pre> apiVersion: v1 kind: Pod metadata: name: mariadb-pod labels: app: mariadb tier: back-end spec: volumes: - name: task-pv-storage persistentVolumeClaim: claimName: task-pv-claim containers: - name: mariadb image: mariadb ports: - containerPort: 3306 env: - name: MARIADB_ROOT_PASSWORD valueFrom: secretKeyRef: name: mariadb-secret key: mariadb-root-password volumeMounts: - name: task-pv-storage mountPath: /var/lib/mysql - name: phpmyadmin image: bitnami/phpmyadmin:latest ports: - containerPort: 8080 </pre>

```
env:
  - name: DATABASE_HOST
    valueFrom:
      configMapKeyRef:
        name: mariadb-configmap
        key: database_url
```

Ejecutamos los siguientes comandos para el arranque del contenedor:

```
root$putin:~# kubectl apply -f mariadb-secret.yaml
root$putin:~# kubectl apply -f mariadb-configmap.yaml
root$putin:~# kubectl apply -f mariadb-pod.yaml
```

Una vez ejecutados, podemos observar el resultado consultando el estado de arranque y ejecución de los contenedores, como se muestra en la Figura 4-3.

```
root@putin:~/resources# kubectl get pod -o wide
NAME          READY   STATUS    RESTARTS   AGE      IP            NODE     NOMINATED NODE   READINESS GATES
mariadb-pod   2/2     Running   0           71s      10.244.1.33   biden    <none>            <none>
nginx-phpfpm  2/2     Running   2 (21h ago) 2d22h     10.244.2.14   merkel   <none>            <none>
```

Figura 4-3 Despliegue contenedores nginx y MariaDB. Autoría propia.

Para tener acceso al servidor MariaDB desde el exterior, vamos a crear un servicio donde podamos conectar desde los clientes y el contenedor *nginx* con la base de datos. Mapeamos el puerto *3306* y lo asociamos a la IP del servidor *putin*, de forma que el servicio accesible por el resto de componentes. El fichero tiene la siguiente configuración:

```
/docker/resources/mariadb-service.yaml

apiVersion: v1
kind: Service
metadata:
  name: mariadb-pod
spec:
  selector:
    app: mariadb
    tier: back-end
  ports:
    - port: 3306
      protocol: TCP
      targetPort: 3306
      nodePort: 30014
  selector:
    app: mariadb
  type: LoadBalancer
  externalIPs:
    - 10.130.120.105
```

Aplicamos la configuración:

```
root$putin:~# kubectl apply -f mariadb-service.yaml
```

Los servicios que están operativos en el sistema son los que podemos observar en la Figura 4-4.

```
root@putin:~/resources# kubectl get svc -o wide
NAME                TYPE                CLUSTER-IP          EXTERNAL-IP          PORT(S)              AGE              SELECTOR
kubernetes           ClusterIP            10.96.0.1            <none>                443/TCP               19d              <none>
mariadb-pod          LoadBalancer        10.102.225.210       10.130.120.105       3306:30014/TCP        42h              app=mariadb
nginx-pod             LoadBalancer        10.107.162.46        10.130.120.105       80:30012/TCP          2d22h            app=nginx-phpfpm
```

Figura 4-4 Servicios balanceados nginx y MariaDB. Autoría propia.

4.2 Modelo lógico de datos

El modelo de datos definido es el mostrado en la Figura 4-5.

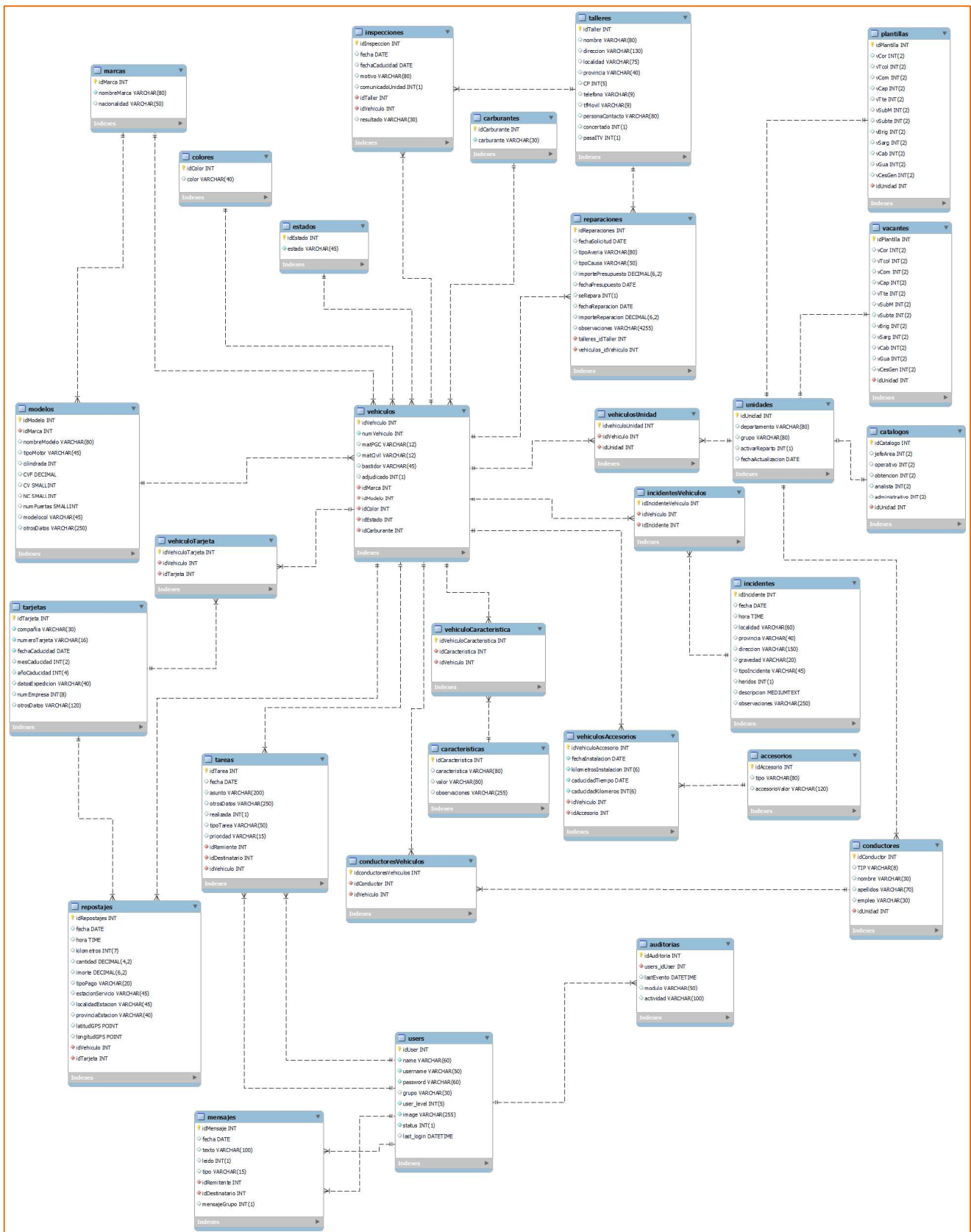


Figura 4-5 Modelo base de datos vehículos. Autoría propia.

Los datos representan el aspecto más importante de cualquier sistema informático y, por tanto, deben construirse sobre una estructura sólida, robusta y precisa.

Los modelos lógicos de datos establecen la estructura de los elementos y datos, así como las relaciones entre ellos. Están compuestos por tres elementos principales:

- *Entidades*: conjunto de elementos relevantes en la aplicación.
- *Relaciones*: asociación entre dos entidades.
- *Atributos*: característica que describe más detalladamente una entidad.

El modelo de datos puede implementarse en cualquier sistema de persistencia de base de datos relacional. Como se ha comentado anteriormente, este proyecto dispone para ello de la base de datos MariaDB.

El diseño que se muestra en la Figura 4-5 está optimizado para albergar la mayor cantidad de información posible acerca de la flota de vehículos de una Unidad, así como para establecer relaciones entre los diferentes elementos definidos, como pueden ser repostajes, incidentes, reparaciones, etc.

Mediante el programa *MySQL Workbench* (entorno gráfico de diseño de base de datos, administración y mantenimiento para *MySQL* y *MariaDB*), creamos las tablas que albergarán la información de la aplicación, como se muestra en la Figura 4-6.

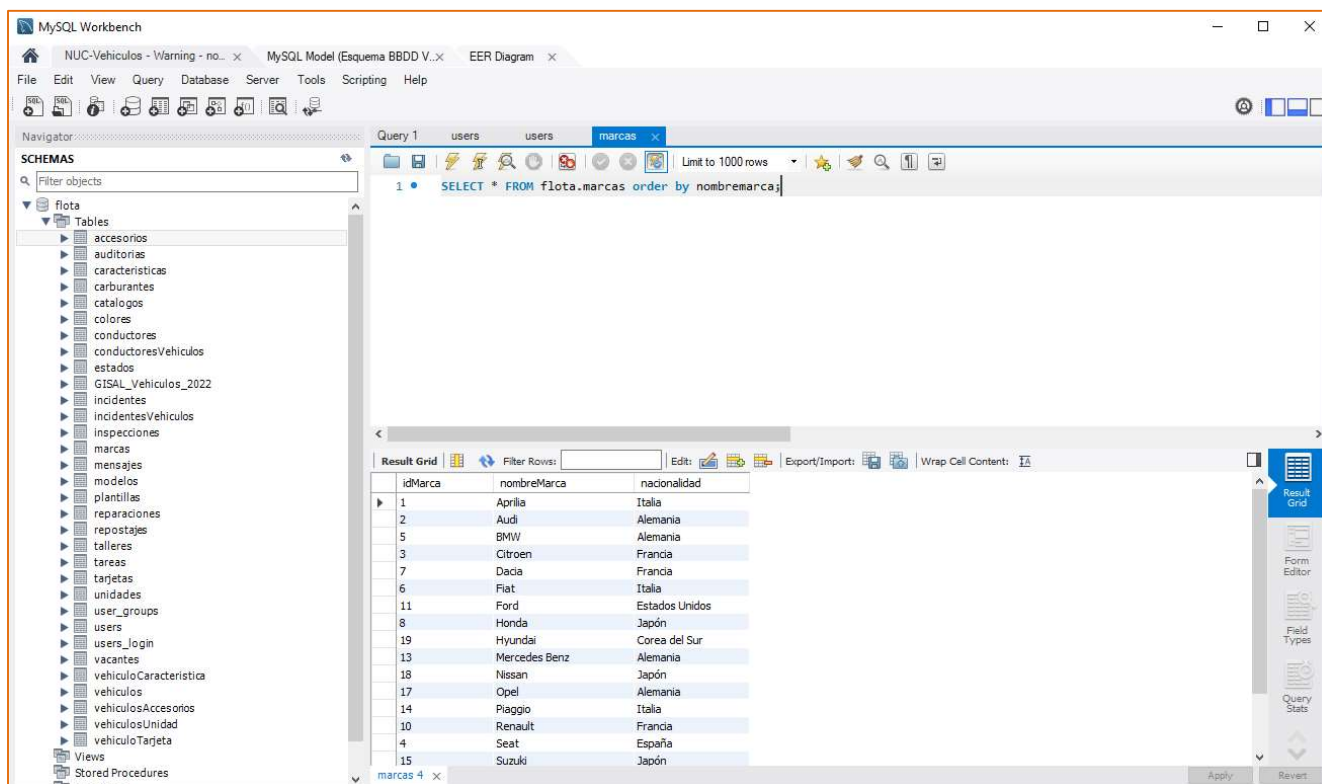


Figura 4-6 Modelo base de datos vehículos. Autoría propia.

4.3 Despliegue de la aplicación

4.3.1 Ficheros PHP de la aplicación y de configuración

Una vez el sistema se encuentra operativo, pasamos a la fase de despliegue de los ficheros, desarrollados en PHP, que forman parte de la aplicación. Como se comentó en el Capítulo 2, los elementos que vamos a utilizar son:

- Bootstrap y jQuery para el diseño, navegación e interacción con las vistas.
- PHP, para el desarrollo y ejecución de los procesos.

Las carpetas, donde se alojan los ficheros de la aplicación, tienen la siguiente estructura:

- *Raíz*. Ficheros PHP de la aplicación.
- *Css*. Hojas de estilo con las que configurar la apariencia de las vistas.
- *Images*. Ubicación de las imágenes de los vehículos que están dados de alta en la aplicación.
- *Img*. Almacena los ficheros de imágenes que se utilizan en las vistas.
- *Includes*. Librerías PHP que contienen funciones para la gestión de la información relacionada con la base de datos y para la ejecución de procedimientos de la aplicación.
- *Js*. Librerías JavaScript de la aplicación.
- *Layouts*. Fragmentos utilizados en la composición de las vistas y que son comunes a varias páginas de la aplicación.

Para que este contenido sea accesible para todos los contenedores *Docker-Nginx*, ubicamos el contenido en el volumen persistente definido como *nfswww*, que está mapeado en la ruta */usr/share/nginx/html* (raíz que sirve el servidor web).

Accedemos a la aplicación, mediante un navegador, alcanzando la página de inicio de sesión del usuario, como se muestra en la Figura 4.7.

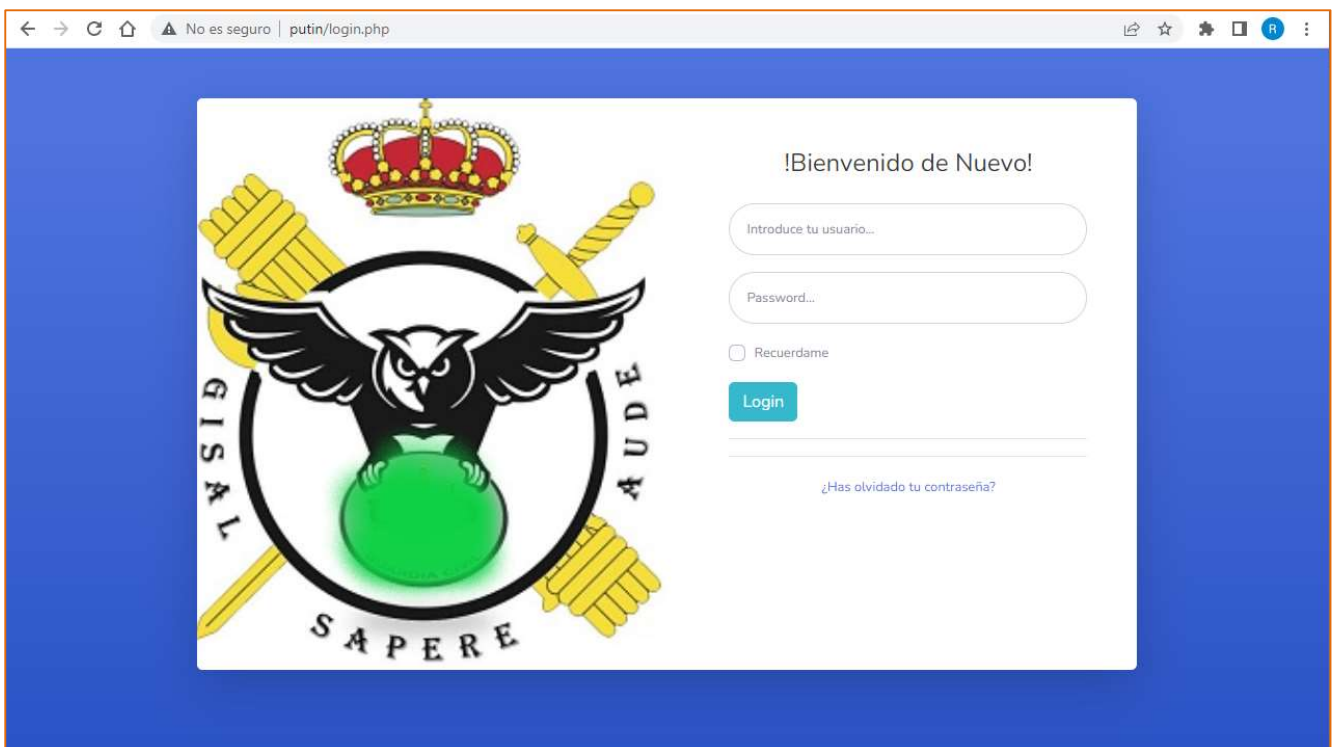


Figura 4-7 Pantalla Login de acceso a la aplicación. Autoría propia.

Cuando se accede a la página de inicio de la aplicación, el sistema comprueba que el usuario no tenga una sesión abierta (almacena información de persistencia y navegación del usuario en la aplicación), como se muestra en la Figura 4-8.

```
<!DOCTYPE html>
<?php
    ob_start();
    require_once('includes/load.php');
    if($session->isUserLoggedIn(true)) { redirect('index.php', false);}
??
```

Figura 4-8 Código PHP registro sesión usuario. Autoría propia.

Si no es así, le solicita el nombre y su contraseña para autenticar su identidad y permitirle a la página principal.

El procedimiento que gestiona la validación del usuario se encuentra en la página denominada *auth.php*, cuyo contenido es el que podemos visualizar en la Figura 4-9.

```
<?php include_once('includes/load.php'); ?>
<?php
$req_fields = array('username','password' );
validate_fields($req_fields);
$username = remove_junk($_POST['username']);
$password = remove_junk($_POST['password']);

if(empty($errors)){
    $user_id = authenticate($username, $password);
    if($user_id){
        //create session with id
        $session->login($user_id);
        //Update Sign in time
        updateLastLogIn($user_id);
        //Añadimos el login del Usuario para tener un histórico
        $_SESSION['user_name'] = $username;
        insertAuditoria($user_id,"Login","Entra al sistema");
        //-----
        $session->msg("s", "Bienvenido al sistema de Gestión de Vehículos -GISAL-.");
        redirect('index.php',false);

    } else {
        $session->msg("d", "Nombre de usuario y/o contraseña incorrecto.");
        redirect('login.php',false);
    }

} else {
    $session->msg("d", $errors);
    redirect('index.php',false);
}

?>
```

Figura 4-9 Validación credenciales usuario. Autoría propia.

Esta página realiza una llamada a la función *authenticate*, con los parámetros usuario y contraseña tecleados por el usuario. Si la información es válida, el sistema registra la entrada en la tabla de auditorías y redirige la ejecución a la página *index.php*.

El código utilizado para el registro de las trazas de auditoría es el que se muestra en la Figura 4-10.

```
/*-----*/
/* Function para añadir registros de "Auditoría" de los usuarios
/*-----*/
function insertAuditoria($user_id, $modulo, $actividad)
{
    global $db;
    $date = make_date();
    $sql = "INSERT INTO auditorias (idUser, lastEvento, modulo, actividad) VALUES ( ";
    $sql .= "'{$user_id}', '{$date}', '{$modulo}', '{$actividad}' )";
    $result = $db->query($sql);
    return ($result && $db->affected_rows() === 1 ? true : false);
}
```

Figura 4-10 Función registro trazas de auditoría. Autoría propia.

Una vez validado el usuario, se accede a la página principal, como se observa en la Figura 4-11.

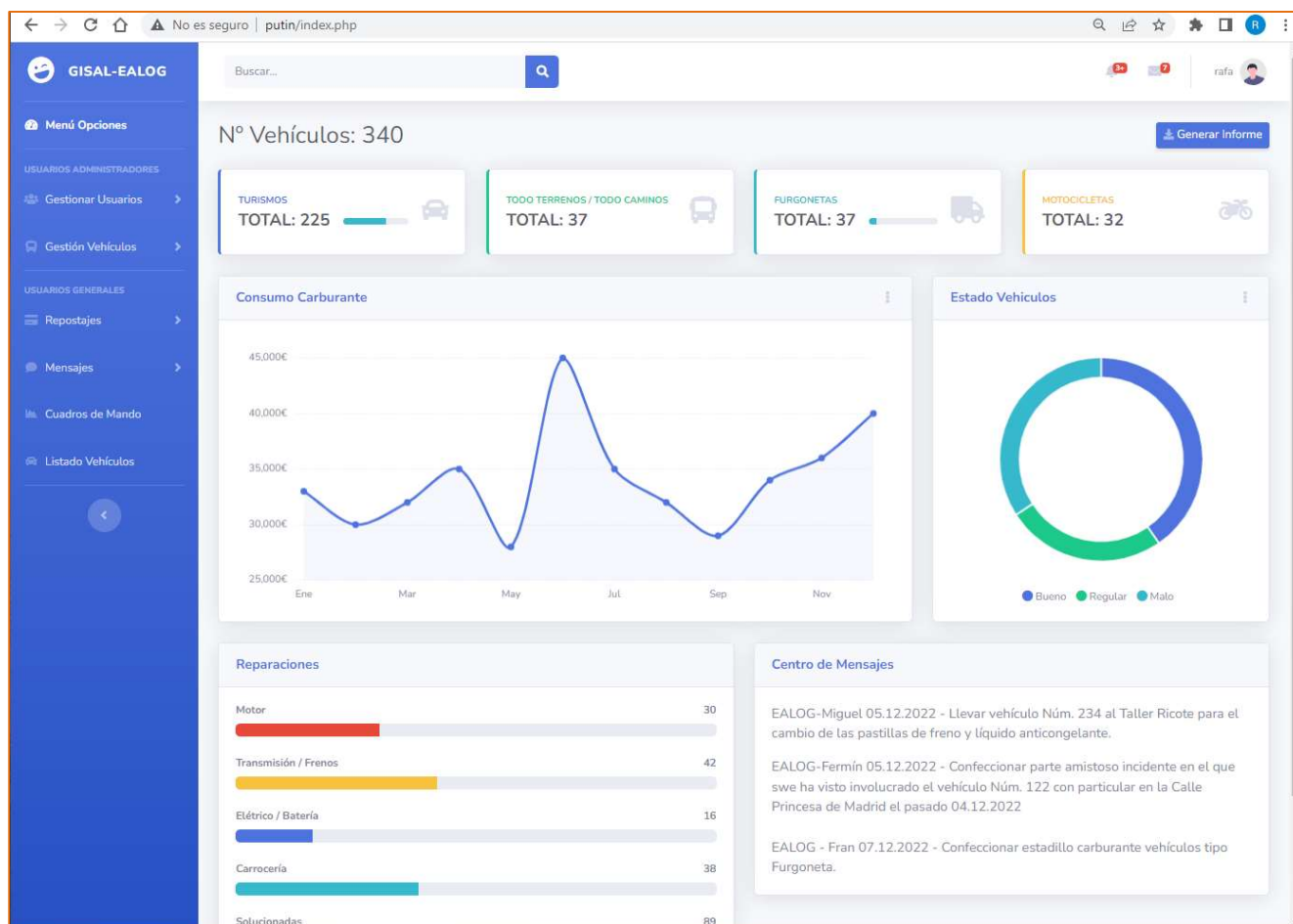


Figura 4-11 Página principal de la aplicación. Autoría propia.

Esta primera página es un *dashboard* que muestra el cuadro de mando principal. En ella se observan diferentes datos resumen del estado actual de la flota de vehículos de la Unidad, como son resúmenes de los tipos de vehículos, estadística de gasto de carburante, estado de los vehículos, número de reparaciones llevadas a cabo durante el año con especificación del tipo de avería y los últimos mensajes recibidos en la aplicación. Está dividida en tres secciones, tal y como se muestra, a continuación, en la Figura 4-12.

- Barra lateral.
 - Incluye el menú principal de navegación de la aplicación.
- Barra Superior.
 - Contiene los siguientes elementos:
 - *Cuadro de búsqueda*: permite la búsqueda de datos relativos a los vehículos de la Unidad.
 - *Centro de alertas*: desplegable que muestra los mensajes de alarma generados por la aplicación y que todavía no han sido validados por los usuarios.
 - *Centro de mensajes*: mensajes enviados por los usuarios al usuario “administrador” que está conectado.
 - *Opciones de usuario*: modificación de datos del perfil de usuario y salida de la aplicación.
- Cuerpo de página.
 - Datos relativos a la opción del menú seleccionada en la aplicación.

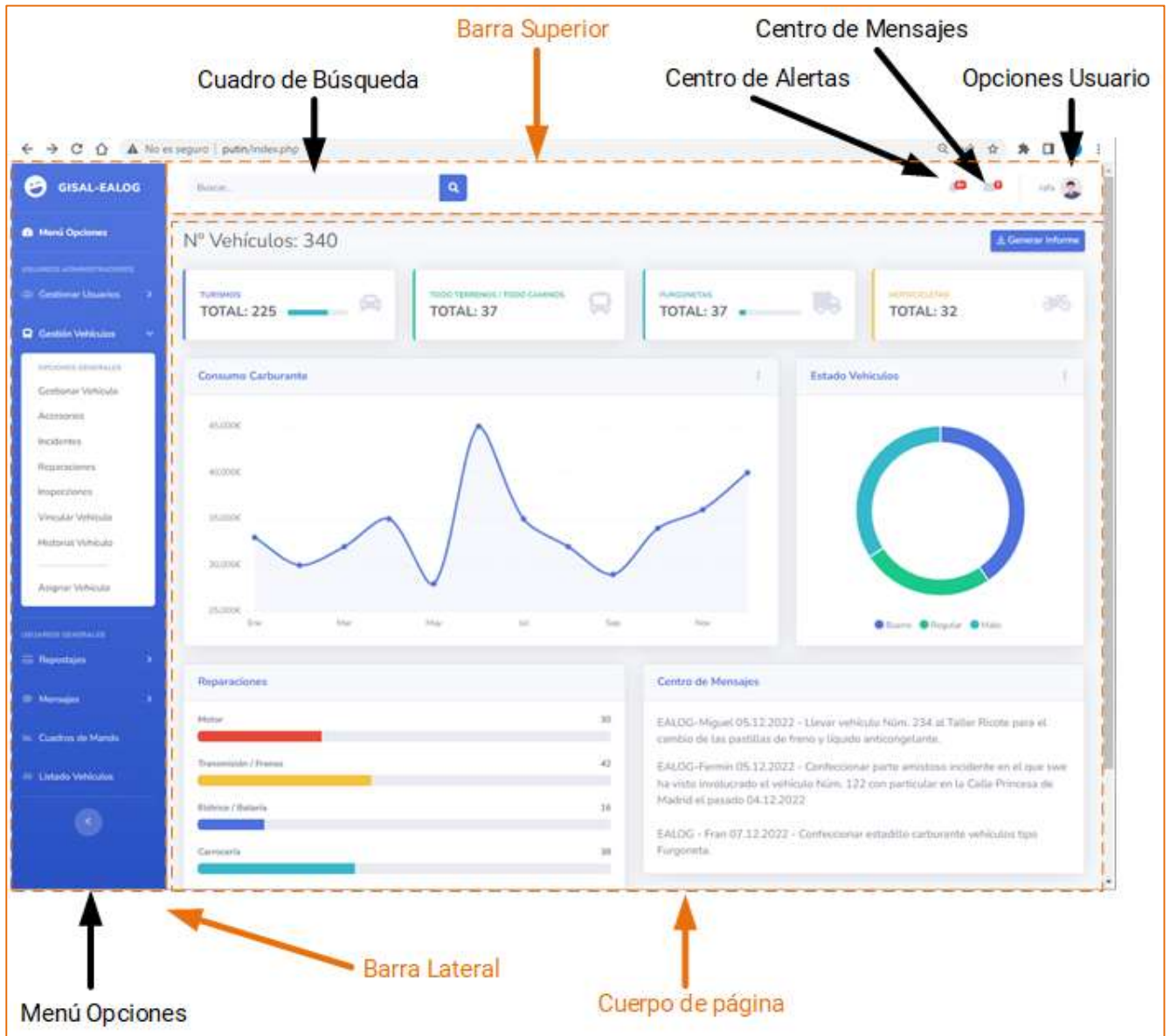


Figura 4-12 Secciones que componen las vistas de la aplicación. Autoría propia.

En esta página principal, en la parte superior, se encuentran las secciones:

- *Centro de alertas*, donde el sistema registra aquellos eventos que deben ser comprobados y administrados por el personal de gestión del Grupo de Apoyo, como, por ejemplo:
 - Vehículos cuyas cubiertas han sobrepasado el kilometraje máximo definido.
 - Vehículos que deben realizar revisiones de motor.
 - Vehículos que deben someterse a una inspección técnica ITV.



Figura 4-13 Mensajes del Centro de Alertas. Autoría propia.

- *Centro de mensajes*, donde se almacenan aquellos que tiene el usuario logueado pendientes de revisar. Estos mensajes son los enviados por los usuarios de la aplicación para solicitar la realización de alguna gestión relativa a los vehículos de su Departamento o Grupo.

Si seleccionamos dentro del Menú Opciones, en el apartado Gestión Vehículos, Gestionar Vehículo, accedemos a la pantalla que se muestra en la Figura 4-14.

Buscar...

Añadir nuevo Vehículo

Número Vehículo: Matricula Interna: Matricula normal: Selecciona una Marca: Selecciona un Modelo:

Número Bastidor: Núm. Kilómetros: Selecciona una Categoría: Selecciona un color:

Selecciona un Estado: Selecciona un Carburante: Selecciona tipo de Vehículo:

Registrar nuevo vehículo

Show 10 entries Search:

Núm.	Tipo	Interna	Matricula	Bastidor	Cat.	Kilómetros	Marca	Modelo	Color	Estado	Carburante
1	Turismo	3615	6851GNL	VF7RDXFVJ9L545219		152052	Citroen	C5	Gris	Regular	Gasolina 95
2	Turismo	8544	6552KLP	VKFF43445333DF		85304	Toyota	Land Cruiser	Azul	Bueno	Gasolina 95
3	Turismo	3585	6515IUL	VUY4434444		13204	BMW	116D	Blanco	Muy Bueno	Diesel
4	Furgon	9321	1258KOP	HYU838383899		43528	Mercedes Benz	Sprinter	Blanco	Regular	Diesel
5	Turismo	8965	3954HYT	OL99939393		98452	Renault	Laguna Grand Tour	Rojo	Regular	Gasolina 95
6	Turismo	5434	8745FRT	VJU7766GYU		242984	Ford	Fiesta 1.4 TDCI	Gris Plata	Malo	Gagolina 98
7	Todotereno	7458	0114JF	VD888383DDD		164528	Mercedes Benz	GLE 350 D	Gris	Bueno	Gagolina 98
8	Turismo	9858	7452MUY	VXJU9393DDD		95054	Mini	Cooper ContryMan	Negro	Muy Bueno	Diesel
9	Furgon	7864	8103GIU	VX999393DD		43052	Renault	Kangoo Express 1.5	Blanco	Regular	Diesel
10	Furgon	8955	7634JUU	VII8993983X		105335	Renault	Kangoo Express 1.5	Negro	Muy Bueno	Gagolina 98
Núm.	Tipo	Interna	Matricula	Bastidor	Cat.	Kilómetros	Marca	Modelo	Color	Estado	Carburante

Showing 1 to 10 of 16 entries Previous 1 2 Next

Figura 4-14 Pantalla de gestión de vehículos. Autoría propia.

En la parte superior se encuentra la sección destinada para dar de alta nuevos vehículos en la flota y, en la parte inferior, figura un listado con todos los vehículos que están activos en la Unidad.

4.3.2 Procedimiento de clasificación de los vehículos

Una de los procedimientos que se pretende automatizar en la aplicación, es la clasificación automática de los vehículos que se dan de alta en el sistema.

Para realizar la clasificación se han tenido en cuenta los siguientes criterios:

- *Tipo de vehículo*. Dependiendo del tipo de vehículo de que se trate, se encuadra su correspondiente categoría.
- *Color*. El color de un vehículo es importante en cuanto aporta discreción y anonimato a la hora de su utilización en actividades de la Unidad.
- *Estado*. Los administradores de la Unidad, cuando reciben el vehículo, efectúan un estudio detallado de su aspecto y estado de conservación, examinando tanto el habitáculo, como la carrocería, motor, kilometraje y características técnicas del mismo, realizando una evaluación y asignándole, temporalmente, una de las categorías que figuran en la Tabla 4-1.
- *Kilómetros actuales*. Se establecen tres intervalos para cada una de las categorías establecidas. La acumulación de kilómetros es un factor que contribuye al desgaste y a la pérdida de fiabilidad del motor, así como a la seguridad del vehículo.

- *Cilindrada*. Los vehículos de alta gama suelen estar dotados de motores de gran cilindrada. Es por ello que, a mayor cilindrada, mayor sea la categoría del vehículo. También se han definido tres tramos de catalogación.
- *CV*. Los caballos de vapor están asociados a la potencia del motor. Es la medición que se ha venido utilizando desde hace bastante tiempo y que ha sido reemplazada, recientemente, por los kilovatios.
- *KW*. Es la medida actual con la que se mide la potencia de un motor (kilovatio). Tras un cambio de normativa, todos los fabricantes tienen la obligación de potenciar el motor con esta medida y es la que, actualmente, es referencia para valorar la fuerza y empuje.

Cat	Tipo Veh.	Color	Estado	Kms.	Cilindrada	CV	KW
A	Turismo	Gris	Muy Bueno	0-80.000	0-1.000	0-75	0-85
	Todoterreno	Gris Plata					
	Todo camino	Aluminio					
	Monovolumen	Blanco					
B	Furgoneta	Naranja	Regular	80.001-150.000	1.001-1.800	76-120	86-130
	Motocicleta	Verde					
	Furgoneta	Verde Osc.					
	Especial	Azul					
C	Camión Remolque	Negro	Malo Muy Malo	150.001-600.000	1.801-10.000	121-900	131-900
		Rojos					
		Verde claro					
		Amarillo					

Tabla 4-1 Baremos de clasificación de los vehículos. Autoría propia.

Cada vez que se agrega un nuevo vehículo, este queda pendiente de clasificación. Para inscribirlo en la categoría correspondiente, se utiliza un procedimiento automático que analiza cada uno de los aspectos descritos en la Tabla 4-1 y, en función del resultado obtenido, lo posiciona en una de las tres especificadas.

El procedimiento está dividido en dos funciones, cuyo código fuente se encuentra en el Anexo I. Las operaciones que llevan a cabo cada una de ellas, son las siguientes:

1. Procesa_caracteristicas_vehiculo

- Encargado de comprobar cada uno de los aspectos de clasificación y almacenar el resultado, de cada uno de ellos, en la tabla *categoríaTemporal*.

b. Categoriza_vehiculos

- Una vez se ha recopilado toda la información, este procedimiento comprueba la calificación obtenida en cada una de las características evaluadas y, en función del resultado, asigna la categoría correspondiente. Para ello, el sistema cuenta la cantidad de puntuaciones que se obtiene en cada una de las categorías (en total son siete los elementos calificados)
- Los parámetros y calificaciones de evaluación con los que se asigna una categoría son los que se muestran en la Tabla 4-2.

Cat. A	Cat. B	Cat. C	Categoría Asignada
6	---	---	A
5	≥ 1	---	A
4	≥ 2	---	B
3	3	---	B
---	≥ 4	---	B
4	---	≥ 2	B
4	$= 2$	> 1	C
---	≥ 3	≥ 2	C
---	---	≥ 3	C

Tabla 4-2 Tabla de clasificación de categorías.

Para comprobar el funcionamiento del sistema, se han introducido 17 nuevos vehículos que cuentan con diferentes características. En el momento de su inclusión en el sistema, se les asigna la categoría “Z”, lo que indica que todavía no están clasificados. El listado de éstos es el que puede observarse en la Figura 4-15.

The screenshot shows a web application interface for vehicle management. On the left is a sidebar menu with options like 'Gestionar Usuarios', 'Gestión Vehículos', and 'Listado Vehículos'. The main area displays a table titled 'Ver listado vehículos por Categoría SIN ASIGNAR'. The table has columns for vehicle details and a 'Cat.' column where all entries are 'Z'. Above the table are buttons for 'Clasifica Vehículos nuevos' and 'Adjudica Vehículos Clasificados'.

Núm.	Tipo	Reservada	Bastidor	Cat.	Kilómetros	Marca	Modelo	Color	Estado	Carburante
1	Turismo	3615	6851GNL	Z	152052	Citroen	C5	Gris	Regular	Gasolina 95
2	Turismo	8544	6552KLP	Z	85304	Toyota	Land Cruiser	Azul	Bueno	Gasolina 95
3	Turismo	3585	6515JUL	Z	13204	BMW	116D	Blanco	Muy Bueno	Diesel
4	Furgon	9321	1258KOP	Z	43528	Mercedes Benz	Sprinter	Blanco	Regular	Diesel
5	Turismo	8965	3954HYT	Z	98452	Renault	Laguna Grand Tour	Rojo	Regular	Gasolina 95
6	Turismo	5434	8745FRT	Z	242984	Ford	Fiesta 1.4 TDCI	Gris Plata	Malo	Gasolina 98
7	Todoterreno	7458	0114JF	Z	164528	Mercedes Benz	GLE 350 D	Gris	Bueno	Gasolina 98
8	Turismo	9858	7452MUY	Z	95054	Mini	Cooper ContryMan	Negro	Muy Bueno	Diesel
9	Furgon	7864	8103GIU	Z	43052	Renault	Kangoo Express 1.5	Blanco	Regular	Diesel
10	Furgon	8955	7634JIU	Z	105335	Renault	Kangoo Express 1.5	Negro	Muy Bueno	Gasolina 98
11	Turismo	4335	1986LLK	Z	12544	Ford	Mondeo HEV Hibrid.	Gris	Muy Bueno	Hibrido - Gasolina
12	Turismo	4336	1987LLK	Z	13525	Ford	Mondeo HEV Hibrid.	Gris	Muy Bueno	Hibrido - Gasolina
13	Turismo	6574	2585LMJ	Z	8745	Ford	Mondeo HEV Hibrid.	Gris	Muy Bueno	Hibrido - Gasolina
14	Turismo	8958	2585LKI	Z	38456	BMW	X3 Drive	Azul	Muy Bueno	Hibrido - diesel
15	Turismo	8574	3788KHY	Z	6587	Hyundai	Kona	Gris	Muy Bueno	Electrico
16	Turismo	1488	5699JLP	Z	98635	Mercedes Benz	GLA 200D	Gris Plata	Bueno	Diesel
17	Turismo	6332	8448MIH	Z	28000	Mercedes Benz	GLE 350 D	Gris Plata	Muy Bueno	Diesel

Figura 4-15 Nuevos vehículos introducidos en el sistema. Autoría propia.

Desde el menú principal, accedemos a la opción *Gestión Vehículos* y posteriormente seleccionamos *Asignar Vehículo*. Como se puede identificar, la categoría inicial de un vehículo, cuando se registra en el sistema, es “Z” (sin catalogar).

Desde la pantalla que se muestra en la Figura 4-15, seleccionamos la opción *Clasifica vehículos nuevos*, procediendo el sistema a realizar la clasificación explicada anteriormente y mostrar el resultado

final, con la categoría asignada a cada uno de los vehículos. Una vez se ha llevado a cabo el procedimiento, observaremos una pantalla como la que se observa en la Figura 4-16.

Núm. T1	Tipo T1	PGC T1	Reservada T1	Bastidor T1	Cat. T1	Kilómetros T1	Marca T1	Modelo T1	Color T1	Estado T1	Carburante T1
1	Turismo	3615	6851GNL	VF7RDXFVJ9L545219	A	152052	Citroen	C5	Gris	Regular	Gasolina 95
2	Turismo	8544	6552KLP	VKFF43445333DF	A	85304	Toyota	Land Cruiser	Azul	Bueno	Gasolina 95
3	Turismo	3585	6515JUL	VUY4434444	C	13204	BMW	116D	Blanco	Muy Bueno	Diesel
4	Furgon	9321	1258KOP	HYU838383899	B	43528	Mercedes Benz	Sprinter	Blanco	Regular	Diesel
5	Turismo	8965	3954HYT	OL99939393	B	98452	Renault	Laguna Grand Tour	Rojo	Regular	Gasolina 95
6	Turismo	5434	8745FRT	VJU7766GYU	C	242984	Ford	Fiesta 1.4 TDCI	Gris Plata	Malo	Gasolina 98
7	Todotereno	7458	0114JIF	VD888383DDD	A	164528	Mercedes Benz	GLE 350 D	Gris	Bueno	Gasolina 98
8	Turismo	9858	7452MUY	VXJU93939DD	B	95054	Mini	Cooper ContryMan	Negro	Muy Bueno	Diesel
9	Furgon	7864	8103GU	VX99393DD	C	43052	Renault	Kangoo Express 1.5	Blanco	Regular	Diesel
10	Furgon	8955	7634JU	VII8993983X	B	105335	Renault	Kangoo Express 1.5	Negro	Muy Bueno	Gasolina 98
11	Turismo	4335	1986LLK	X773D44	A	12544	Ford	Mondeo HEV Hibrid.	Gris	Muy Bueno	Hibrido - Gasolina
12	Turismo	4336	1987LLK	X884DDHJ6634	A	13525	Ford	Mondeo HEV Hibrid.	Gris	Muy Bueno	Hibrido - Gasolina
13	Turismo	6574	2585LMJ	X8938939DD	A	8745	Ford	Mondeo HEV Hibrid.	Gris	Muy Bueno	Hibrido - Gasolina
14	Turismo	8958	2585LKI	VIOL334433LOLL	A	38456	BMW	X3 Drive	Azul	Muy Bueno	Hibrido - diesel
15	Turismo	8574	3788KHY	VIOL334433LOLL3444	B	6587	Hyundai	Kona	Gris	Muy Bueno	Electrico
16	Turismo	1488	5699JLP	VJ88348883DD33	A	98635	Mercedes Benz	GLA 200D	Gris Plata	Bueno	Diesel
17	Turismo	6332	8448MJH	V89FF66774343DCC	A	28000	Mercedes Benz	GLE 350 D	Gris Plata	Muy Bueno	Diesel

Figura 4-16 Clasificación de los nuevos vehículos. Autoría propia.

Analizamos el resultado de la clasificación y de la puntuación obtenida en cada uno de los parámetros examinados por el sistema para cercionarnos que, efectivamente, el procedimiento funciona correctamente.

Nº	Tipo	Cilin.	CV	KW	kms.	Marca	Modelo	Color	Estado
1	Turismo	2.946	210	155	152052	Citroen	C5	Gris	Regular
A	A	A	A	A	C			A	B
2	Turismo	2.755	204	150	85304	Toyota	Land Cruiser	Azul	Bueno
A	A	A	A	A	B			B	A
3	Turismo	1.496	115	85	13204	BMW	116D	Blanco	Muy Bueno
C	A	B	B	C	A			A	A
4	Furgon	2.143	114	84	43528	Mercedes Benz	Sprinter	Blanco	Regular
B	B	A	B	C	A			A	B
5	Turismo	1.995	149	110	98452	Renault	Laguna Grand Tour	Rojo	Regular
B	A	A	A	B	B			C	B
6	Turismo	1.753	115	85	242984	Ford	Fiesta 1.4 TDCI	Gris Plata	Malo
C	A	B	B	C	C			A	C
7	Todotereno	2.987	258	190	164528	Mercedes Benz	GLE 350 D	Gris	Bueno
A	A	A	A	A	C			A	A
8	Turismo	1.499	136	100	95054	Mini	Cooper ContryMan	Negro	Muy Bueno
B	A	B	A	B	B			B	A
9	Furgón	1.461	65	48	43052	Renault	Kangoo Express 1.5	Blanco	Regular

C	B	B	C	C	A			A	B
10	Furgón	1.461	65	48	105335	Renault	Kangoo Express 1.5	Negro	Muy Bueno
B	B	B	C	C	B			B	A
11	Turismo	1.999	140	103	12544	Ford	Mondeo HEV Hibrid.	Gris	Muy Bueno
A	A	A	A	B	A			A	A
12	Turismo	1.999	140	103	13525	Ford	Mondeo HEV Hibrid.	Gris	Muy Bueno
A	A	A	A	B	A			A	A
13	Turismo	1.999	140	103	8745	Ford	Mondeo HEV Hibrid.	Gris	Muy Bueno
A	A	A	A	B	A			A	A
14	Turismo	1.995	183	135	38456	BMW	X3 Drive	Azul	Muy Bueno
A	A	A	A	A	A			B	A
15	Turismo	998	89	66	6587	Hyundai	Kona	Gris	Muy Bueno
B	A	C	B	C	A			A	A
16	Turismo	2.143	136	100	98635	Mercedes Benz	GLA 200D	Gris Plata	Bueno
A	A	A	A	B	B			A	A
17	Turismo	2.987	258	190	28000	Mercedes Benz	GLE 350 D	Gris Plata	Muy Bueno
A	A	A	A	A	B			A	A

Tabla 4-3 Tabla comprobación procedimiento de clasificación. Autoría propia.

Como se observa en la tabla comparativa, los resultados se ajustan a los parámetros definidos en el procedimiento de clasificación.

4.3.3 Procedimiento de adjudicación de los vehículos

Este procedimiento es el *alma máter* de la aplicación, lo que le ha dado vida e impulso para su diseño y programación.

Anteriormente, la adjudicación de los vehículos se realizaba directamente por el responsable del Departamento de Apoyo Logístico, apoyándose en criterios y procedimientos que podrían ser considerados, por el resto del personal, como “subjetivos”, por lo que nunca tuvo la aprobación y verificación de los componentes de la Unidad. En este proyecto se lleva a cabo el desarrollo de un procedimiento con el que poder adjudicar, de forma automática, los vehículos pendientes de asignación, apoyándose en parámetros tasados, públicos y que valoren los aspectos de plantilla, vacantes y puestos de trabajo definidos en la Unidad.

En este proyecto, se ha definido una Unidad con la siguiente estructura orgánica compuesta por:

- 5 Departamentos
- 12 Grupos

Se toma como unidad base, para realizar el reparto, al “Grupo”, dado que es la Unidad base que concentra otras unidades de nivel inferior, como son las secciones, gabinets y equipos.

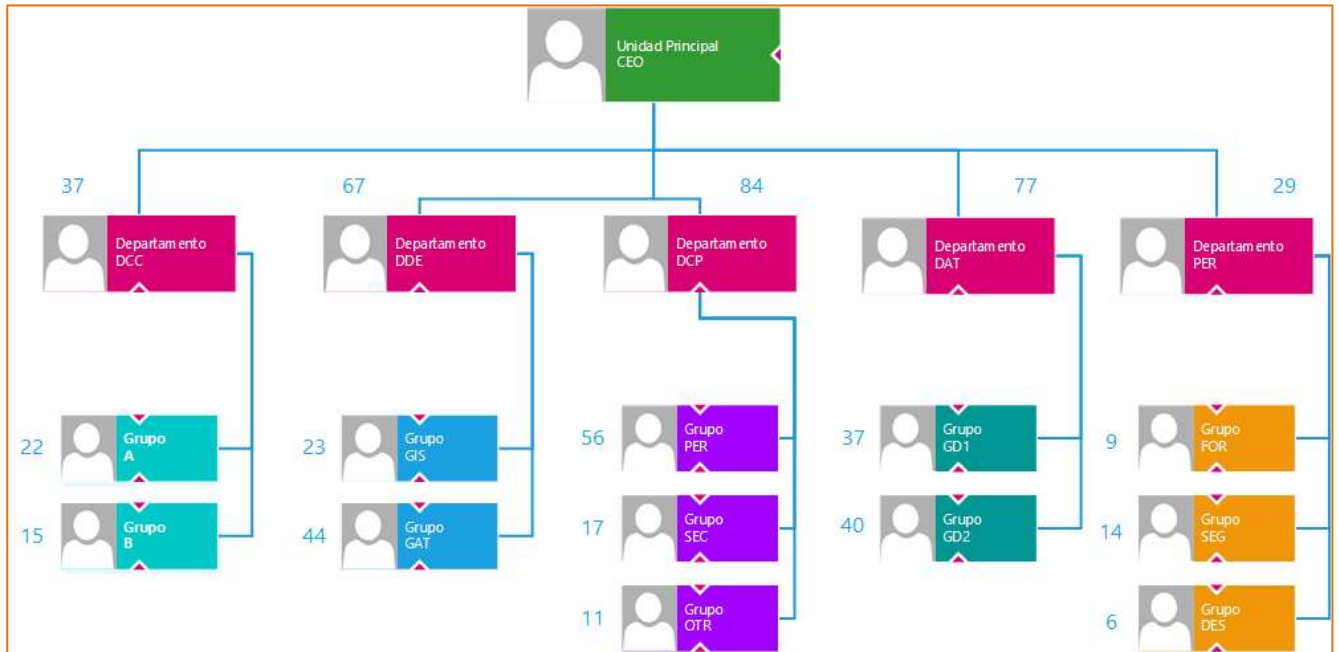


Figura 4-17 Organigrama de organización de la Unidad. Autoría propia.

Como se puede observar en la Figura 4-17, la Unidad cuenta con un total de 294 puestos de trabajo, repartidos en 5 Departamentos.

El catálogo de los puestos de trabajo que se han definido en este proyecto son los que figuran en la Tabla 4-4.

Depart.	Grupo													
Plantilla		CEO	CIO	Jef. G.	Jef. Sec.	Direct.	Cast.	Lab.	Sublab.	Per.	Técnico	Oficina	Apoyo	Total
DCC	A			1	1			1	2	4	2	9	2	22
DCC	B			1					1	3	2	8		15
DAT	GAT			1	1	1	2	2	4	2	6	20	5	44
DAT	GIS			1	1	1			1	2	2	11	4	23
DCP	PER		1	1	2	1	3	3	2	6	8	27	2	56
DCP	SEC			1		1		1		3	2	8	1	17
DCP	OTR				1					2	1	7		11
DDE	GD1		1	1		1	1	2	2	5	2	20	2	37
DDE	GD2			1	1			2	2	6	6	18	4	40
PER	FOR			1						1	1	4	2	9
PER	SEG												14	14
PER	DES			1			1					2	2	6
														294

Tabla 4-4 Tabla Catálogo puestos de trabajo de la Unidad. Autoría propia.

Lo aconsejable en esta Unidad es que la plantilla estuviese ocupada al 100%, pero, por norma general, los Departamentos y Grupos suelen tener puestos vacantes pendientes de asignación.

El factor más importante a la hora de evaluar la prioridad en la asignación, así como establecer el número recomendado de los vehículos totales que pueden asignarse a un Grupo, es la puntuación que se estipula en función del puesto de trabajo. Esta catalogación se define por parte de la Unidad y, en el caso de este proyecto, se han definido cinco tipos distintos.

Cada Departamento y Grupo divide, la plantilla que tiene adjudicada, en estos cinco tipos dependiendo de las labores que desempeña cada componente. Como puede observarse en la Tabla 4-5, los puestos definidos en cada Departamento y Grupo son los siguientes:

Depart.	Grupo						
Catálogo		Jef. -Área	Oper.	Obten.	Analista	Administ.	TOTAL
DCC	A	2		8	10	2	22
DCC	B	1	12	2			15
DAT	GAT	3	26	8	2	5	44
DAT	GIS	3	6	4	6	4	23
DCP	PER	5	35		14	2	56
DCP	SEC	2	2	6	6	1	17
DCP	OTR	1	4	3	2	1	11
DDE	GD1	3	5	12	15	2	37
DDE	GD2	2	8	16	10	4	40
PER	FOR	1	6			2	9
PER	SEG					14	14
PER	DES	1	3		2		6
							294

Tabla 4-5 Tabla Definición puestos de trabajo. Autoría propia.

Dependiendo del cometido y función que se designa a cada puesto de trabajo, se estipula una puntuación con la que poder calcular el porcentaje en la prioridad para la asignación de un nuevo vehículo.

La puntuación asignada a estos puestos de trabajo es la que figura en la Tabla 4-6. Estos valores son los que define la Junta Directiva y pueden ser modificados en cualquier momento.

	Vehículo	Jef. -Área	Oper.	Obten.	Analista	Administ.
Puntuación	A	1	0,60	0,50	0,30	0,20
	B	0	0,60	0,50	0,30	0,20
	C	0	0,40	0,50	0,30	0,80

Tabla 4-6 Puntuación puestos del catálogo. Autoría propia.

Como puede observarse en la Tabla 4-6, dependiendo de la clasificación de los vehículos que se explicó en el apartado 4.3.2 (Categoría A, B y C), la puntuación para la adjudicación de cada uno de ellos varía en función del puesto de trabajo definido.

Por ejemplo, en la Tabla 4-6 se detalla que el Jefe de Área tiene un valor de “1”, lo que significa que, por cada puesto de trabajo de este tipo, debe tener adjudicado un vehículo. Si en un Grupo existen dos puestos de Jefe de Área, a este Grupo, en esta categoría, le corresponden dos vehículos.

La fórmula para calcular el porcentaje que corresponde a cada uno de los Grupos de la Unidad es la siguiente:

$$\text{Porcentaje} = \left(\frac{(\text{Jef.Área} \cdot \text{Núm-vac}) + (\text{Oper} \cdot \text{Núm-vac}) + (\text{Obten.} \cdot \text{Núm-vac}) + (\text{Analista} \cdot \text{Núm-vac}) + (\text{Adm.} \cdot \text{Núm-vac})}{\text{Total puestos del Catálogo}} \right)$$

Se selecciona el número de puestos de trabajo que tiene establecido un Grupo del catálogo de la Unidad; se restan las vacantes que tiene actualmente esta categoría y, por último, el resultado se multiplica por el factor de puntuación del puesto de trabajo.

Realizamos una simulación respecto a los vehículos de categoría “A”, y obtenemos los datos que figuran en la Tabla 4-7.

Cat. A								
Depart.	Grupo	Jef. -Área	Oper.	Obten.	Analista	Administ.	Suma	Porcentaje
DCC	A	1,00	0,00	3,50	2,70	0,40	7,60	6,01
DCC	B	1,00	6,00	0,50	0,00	0,00	7,50	5,93
DAT	GAT	3,00	15,60	4,00	0,60	1,00	24,20	19,15
DAT	GIS	2,00	3,60	2,00	1,80	0,60	10,00	7,91
DCP	PER	4,00	16,20	0,00	3,60	0,40	24,20	19,15
DCP	SEC	2,00	1,20	2,50	1,20	0,00	6,90	5,46
DCP	OTR	1,00	2,40	1,50	0,60	0,20	5,70	4,51
DDE	GD1	3,00	2,40	4,00	3,90	0,00	13,30	10,52
DDE	GD2	2,00	4,20	7,00	2,70	0,60	16,50	13,05
PER	FOR	1,00	3,60	0,00	0,00	0,20	4,80	3,80
PER	SEG	0,00	0,00	0,00	0,00	2,60	2,60	2,06
PER	DES	1,00	1,80	0,00	0,30	0,00	3,10	2,45

Tabla 4-7 Simulación cálculo porcentaje reparto vehículos categoría “A”. Autoría propia.

Observando estos datos, podemos advertir que los Grupos que obtendrían la primera adjudicación de un vehículo de categoría “A” serían los que han obtenido un porcentaje de **19,15%**: Departamento **DAT**, Grupo **GAT** y Departamento **DCP**, Grupo **PER**.

Pero, para establecer el valor final de adjudicación, necesitamos tener en cuenta los vehículos de la categoría “A” que tienen adjudicados, actualmente, los Grupos de la Unidad. Realizamos una operación de búsqueda en la base de datos y calculamos el porcentaje que ello supone en el total de la Unidad.

		Núm. Veh.	% Catálogo	% Vehículos	% Final
Porcentaje Final		Cat. A	Cat. A	Cat. A	Cat. A
DCC	A	1	6,01	5,00	1,01
DCC	B	1	5,93	5,00	0,93
DAT	GAT	4	19,15	20,00	-0,85
DAT	GIS	1	7,91	5,00	2,91
DCP	PER	4	19,15	20,00	-0,85
DCP	SEC	1	5,46	5,00	0,46
DCP	OTR	1	4,51	5,00	-0,49
DDE	GD1	2	10,52	10,00	0,52
DDE	GD2	2	13,05	10,00	3,05
PER	FOR	1	3,80	5,00	-1,20
PER	SEG	1	2,06	5,00	-2,94
PER	DES	1	2,45	5,00	-2,55

Tabla 4-8 Cálculo porcentaje final para adjudicación vehículos categoría “A”. Autoría propia.

Como se detalla en la Tabla 4-8, para obtener el porcentaje final restamos, del porcentaje del catálogo de cada Grupo, el porcentaje de los vehículos que tiene adjudicados.

A la vista de la información que figuran en la columna “% Final” de la Tabla 4-8, el Grupo al que habría que adjudicar un nuevo vehículo de la categoría “A”, puesto que es el que tiene un porcentaje mayor, correspondería a: Departamento **DDE**, Grupo **GD2**.

Llevamos esta lógica de proceso al sistema, realizando un procedimiento cuyo código figura en el Anexo II de este Proyecto. Partimos de una Unidad que no tiene ningún vehículo adjudicado y, recientemente, ha incorporado a su parque móvil, un total de diecisiete vehículos, clasificados en tres categorías (A, B y C).

El listado de los vehículos es el que se muestra en la Figura 4-15 (previo a su clasificación) y en la Figura 4-16 (una vez clasificados).

Seleccionamos, en el menú principal, la opción *Gestión Vehículos -> Asignar Vehículo*, mostrándonos una pantalla con los vehículos de la flota que actualmente están pendientes de asignación.

De los diecisiete vehículos, nueve pertenecen a la categoría A, cinco a la categoría B y tres a la categoría C. Pulsamos en el botón *Adjudica Vehículos Clasificados* y, el resultado final, es el que se muestra en la Figura 4-18.

Núm.	Tipo	Matrícula	Cat.	Kilómetros	Marca	Modelo	Color	Estado	Dpto.	Grupo
1	Turismo	6851GNL	A	152052	Citroen	C5	Gris	Regular	Dpto. DAT	Grupo GAT
2	Turismo	6552KLP	A	85304	Toyota	Land Cruiser	Azul	Bueno	Dpto. DCP	Grupo PER
7	Todotereno	0114UUF	A	164528	Mercedes Benz	GLE 350 D	Gris	Bueno	Dpto. DDE	Grupo GD2
11	Turismo	1986LLK	A	12544	Ford	Mondeo HEV Hibrid.	Gris	Muy Bueno	Dpto. DDE	Grupo GD1
12	Turismo	1987LLK	A	13525	Ford	Mondeo HEV Hibrid.	Gris	Muy Bueno	Dpto. DAT	Grupo GIS
13	Turismo	2585LMJ	A	8745	Ford	Mondeo HEV Hibrid.	Gris	Muy Bueno	Dpto. DCC	Grupo A.
14	Turismo	2585LKI	A	38456	BMW	X3 Drive	Azul	Muy Bueno	Dpto. DCC	Grupo B.
16	Turismo	5699JLP	A	98635	Mercedes Benz	GLA 200D	Gris Plata	Bueno	Dpto. DCP	Grupo SEC
17	Turismo	8448MJH	A	28000	Mercedes Benz	GLE 350 D	Gris Plata	Muy Bueno	Dpto. DAT	Grupo GAT
4	Furgon	1258KOP	B	43528	Mercedes Benz	Sprinter	Blanco	Regular	Dpto. DAT	Grupo GAT
5	Turismo	3954HYT	B	98452	Renault	Laguna Grand Tour	Rojo	Regular	Dpto. DCP	Grupo PER
8	Turismo	7452MUY	B	95054	Mini	Cooper ContryMan	Negro	Muy Bueno	Dpto. DDE	Grupo GD2
10	Furgon	7634JJU	B	105335	Renault	Kangoo Express 1.5	Negro	Muy Bueno	Dpto. DDE	Grupo GD1
15	Turismo	3788KHY	B	6587	Hyundai	Kona	Gris	Muy Bueno	Dpto. DAT	Grupo GIS
3	Turismo	6515JUL	C	13204	BMW	116D	Blanco	Muy Bueno	Dpto. DAT	Grupo GAT
6	Turismo	8745FRT	C	242984	Ford	Fiesta 1.4 TDCI	Gris Plata	Malo	Dpto. DCP	Grupo PER
9	Furgon	8103GJU	C	43052	Renault	Kangoo Express 1.5	Blanco	Regular	Dpto. DDE	Grupo GD2

Figura 4-18 Resultado final de la adjudicación de los vehículos. Autoría propia.

Como puede observarse en la Figura 4-18, varios grupos han sido agraciados con varios vehículos de una misma categoría, mientras otros no han recibido ninguno. Ello es debido a que cada vez que se adjudica un vehículo, el porcentaje de adjudicación de vehículos varía, al igual que el porcentaje de adjudicación.

Realizamos una simulación, en una hoja de cálculo, para comprobar el funcionamiento, por ejemplo, de los vehículos de la categoría “A”.

Como se mencionó anteriormente, el Grupo al que se adjudica el primer vehículo es aquel cuyo porcentaje de Catálogo es el más alto del total de la Unidad. Como se muestra en la Tabla 4-7, el valor más alto corresponde al Dpto. **DAT**, Grupo **GAT**. Una vez se adjudica, se vuelven a sumar los vehículos adjudicados de una categoría para cada Grupo y se establecen nuevos valores de porcentajes finales, como puede observarse en la Tabla 4-9.

		Núm. Veh.	% Catálogo	% Vehículos	% Final
Porcentaje Final		Cat. A	Cat. A	Cat. A	Cat. A
DCC	A	0	6,01	0,00	6,01
DCC	B	0	5,93	0,00	5,93
DAT	GAT	1	19,15	100,00	-80,85
DAT	GIS	0	7,91	0,00	7,91
DCP	PER	0	19,15	0,00	19,15
DCP	SEC	0	5,46	0,00	5,46
DCP	OTR	0	4,51	0,00	4,51
DDE	GD1	0	10,52	0,00	10,52
DDE	GD2	0	13,05	0,00	13,05
PER	FOR	0	3,80	0,00	3,80
PER	SEG	0	2,06	0,00	2,06
PER	DES	0	2,45	0,00	2,45

Tabla 4-9 Adjudicación segundo vehículo categoría “A”. Autoría propia.

A la vista de la información que podemos visualizar en la Tabla 4-9, el siguiente Grupo al que le correspondería la adjudicación del segundo vehículo de la categoría “A”, sería al Dpto. **DCP**, Grupo **PER**, puesto que ostenta la puntuación más alta de las que figuran en la columna “% Final” (19.15%)

Continuamos con la secuencia de adjudicación y, en esta tercera ocasión, el Grupo con mayor puntuación y al que le corresponde la adjudicación del vehículo, es del Dpto. **GDE**, Grupo **GD2**, como puede comprobarse en la Tabla 4-10.

		Núm. Veh.	% Catálogo	% Vehículos	% Final
Porcentaje Final		Cat. A	Cat. A	Cat. A	Cat. A
DCC	A	0	6,01	0,00	6,01
DCC	B	0	5,93	0,00	5,93
DAT	GAT	1	19,15	50,00	-30,85
DAT	GIS	0	7,91	0,00	7,91
DCP	PER	1	19,15	50,00	-30,85
DCP	SEC	0	5,46	0,00	5,46
DCP	OTR	0	4,51	0,00	4,51
DDE	GD1	0	10,52	0,00	10,52
DDE	GD2	0	13,05	0,00	13,05
PER	FOR	0	3,80	0,00	3,80
PER	SEG	0	2,06	0,00	2,06
PER	DES	0	2,45	0,00	2,45

Tabla 4-10 Adjudicación tercer vehículo categoría “A”.

Para comprobar que el noveno vehículo de la categoría “A” le corresponde de nuevo al Grupo **GAT**, del Dpto. **DAT**, pasaremos a visualizar la Tabla 4-11, de porcentajes de reparto, cuando ya se ha efectuado la octava adjudicación.

		Núm. Veh.	% Catálogo	% Vehículos	% Final
Porcentaje Final		Cat. A	Cat. A	Cat. A	Cat. A
DCC	A	1	6,01	12,50	-6,49
DCC	B	1	5,93	12,50	-6,57
DAT	GAT	1	19,15	12,50	6,65
DAT	GIS	1	7,91	12,50	-4,59
DCP	PER	1	19,15	12,50	6,65
DCP	SEC	1	5,46	12,50	-7,04
DCP	OTR	0	4,51	0,00	4,51
DDE	GD1	1	10,52	12,50	-1,98
DDE	GD2	1	13,05	12,50	0,55
PER	FOR	0	3,80	0,00	3,80
PER	SEG	0	2,06	0,00	2,06
PER	DES	0	2,45	0,00	2,45

Tabla 4-11 Tabla porcentajes de reparto para la novena adjudicación. Autoría propia.

Al recalcularse de nuevo los porcentajes, en cuanto al total de vehículos asignados, podemos observar cómo el Grupo **GAT** ostenta el mayor porcentaje en la tabla, por lo que el noveno vehículo le será de nuevo adjudicado.

Por todo ello, podemos afirmar que el procedimiento desarrollado en la aplicación se ajusta a los principios de equidad, neutralidad e integridad, definidos en el punto 1.2 (Motivación del proyecto), lo que garantiza la proporcionalidad e igualdad entre todos los Departamentos y Grupos de la Unidad.

5 CONCLUSIONES Y LÍNEAS FUTURAS

5.1 Conclusiones

El objetivo principal de este proyecto era diseñar un sistema integral, basado en tecnologías de virtualización y contenerización, en el que se pueda ejecutar una aplicación que gestione una flota de vehículos de una Unidad.

Con ello se pretende aumentar la eficiencia y productividad del grupo encargado de dicha gestión, mejorar y optimizar el estado y funcionamiento de los vehículos, así como mostrar una visión global del estado y situación de la flota en tiempo real.

El primer objetivo lo hemos completado diseñando, configurando e instalando un entorno de contenedores Docker, gestionados por Kubernetes. Apoyándonos en una herramienta de configuración para múltiples sistemas mediante la ejecución de *scripts* de Ansible, denominados *playbooks*, hemos desplegado una infraestructura *dockerizada*, compuesta por tres nodos, con la que configurar los distintos elementos que albergarán el sistema. Una vez el nodo *master* tiene comunicación con los nodos *worker*, y el *cluster* se encuentra perfectamente configurado y en ejecución, se ha completado el entorno con el establecimiento de un repositorio persistente de datos, en un volumen NFS, visible para todo el entorno de trabajo.

Posteriormente, y como segundo objetivo expuesto en el proyecto, se ha acometido el diseño de los componentes que componen la aplicación de gestión, definiendo los objetivos que se pretenden alcanzar, estableciendo los requisitos funcionales y no funcionales que debe contemplar el sistema, diseñando el gráfico de actores y casos de uso global del sistema y finalizando con la documentación de los casos de uso especificados en el proyecto.

Por último, se ha abordado el procedimiento de adjudicación de vehículos, diseñando un modelo de negocio, compuesto de varios elementos interrelacionados entre sí, con los que adjudicar un vehículo que se ha incorporado recientemente a uno de los Grupos de la Unidad, todo ello bajo los principios de imparcialidad, transparencia, igualdad y publicidad.

Finalmente, se han programado y validado los procedimientos de clasificación y adjudicación de vehículos, con los que se pretende agilizar, automatizar y optimizar el trabajo que desarrollan los Equipos de Apoyo Logístico en dichas tareas.

En el Capítulo IV, hemos llevado a cabo el despliegue de los contenedores, utilizando imágenes registradas en Docker Hub, configurado un entorno de *front-end* con el servidor web Nginx y con el preprocesador PHP-FPM, encargado de la ejecución del contenido PHP de las páginas de la aplicación y, en la parte de *back-end*, activando el contenedor de la base de datos MariaDB. Todos estos despliegues los hemos llevado a cabo utilizando ficheros YAML de Kubernetes, configurando los contenedores con los parámetros de versión de la aplicación, puertos de comunicación y volúmenes de almacenamiento permanente.

Se ha realizado el diseño de una base de datos relacional, con las entidades, relaciones y atributos que utiliza el sistema para almacenar la información. Se han diseñado y programado varios ficheros en lenguaje PHP y HTML, con los que ofrecer un portal gráfico al usuario. Para ofrecer un mayor dinamismo y diseño, se han utilizado los *frameworks* Bootstrap y jQuery.

Para identificar a los usuarios, se ha desarrollado un procedimiento de autenticación y autorización con el que validar la entrada al sistema. Una vez el usuario se ha identificado, sus datos se registran en el sistema de auditoría y, a hasta el momento de su salida, todas las acciones que realice en el mismo quedarán auditadas.

También se ha desarrollado e implementado un *dashboard* genérico, con el que poder visualizar, de forma rápida y dinámica, el estado actual de toda la flota de vehículos.

Para realizar la clasificación de vehículos, se ha programado un recurso de clasificación con el que realizar esta tarea de forma automática y descentralizada, utilizando un catálogo de parámetros definidos previamente por la Unidad. De esta forma, se descarga de trabajo al Equipo de Apoyo Logístico y se obtiene una clasificación muy rápida de todos los vehículos pendientes de catalogar.

Finalmente, y como desafío principal de este proyecto, se ha desarrollado y programado el procedimiento de adjudicación automática de vehículos a los distintos Grupos que componen la Unidad. Para validar su funcionamiento y comprobar que el código desarrollado se ejecuta correctamente, así como que realiza un reparto equitativo y en función de los parámetros definidos, se ha llevado a cabo una simulación con diecisiete vehículos y doce Grupos de la Unidad, utilizando datos de plantilla, vacantes y catalogación de puestos de trabajo. Se han clasificado los diecisiete vehículos en las tres categorías definidas (A, B y C) y, por último, se ha realizado el reparto de los vehículos de categoría A a los doce Grupos, obteniendo un resultado positivo y correcto en la adjudicación.

Por todo ello, podemos constatar que los objetivos que se han planteado al inicio de este trabajo se han realizado completamente. Convendría completar la aplicación con los módulos, elementos y requisitos definidos en el Capítulo IV, para finalizar la aplicación.

5.2 Líneas futuras

Para aumentar la automatización del sistema y reducir tareas burocráticas del personal de la Unidad, debería dotarse al sistema de una *App*, desarrollada para sistemas Android, que permita a los agentes de la Unidad, registrar cada uno de los repostajes de combustible que lleven a cabo en sus desplazamientos. Los datos serían enviados, vía conexión VPN privada, al sistema central, utilizando tecnología API REST, en cuyo centro de datos (*back-end*) y utilizando procedimientos programados en lenguaje PHP, respaldarían la información en la base de datos de la aplicación. Estos datos, obtenidos en tiempo real, permitirían automatizar valores como son los kilómetros actuales que ha recorrido un vehículo, los repostajes realizados en un periodo de tiempo determinado, cálculos de los tiempos previstos para realizar operaciones de revisión y mantenimientos, etc.

La interfaz de esta App será lo más ligera y sencilla posible para realizar esta actividad y enviar la información al sistema central de la manera más rápida y ágil posible, de forma que los agentes empleen el menor tiempo en realizarla e interfiera lo más mínimo en su trabajo.

Debería desarrollarse un módulo para el almacenamiento y visualización de imágenes y vídeos que puedan realizarse de los distintos medios de transporte, con los que documentar su estado, evolución, reparaciones, etc.

Sería recomendable programar una API que facilite la importación y exportación de datos a otros formatos, como por ejemplo CVS, TXT, JSON, etc., así como con aplicaciones ofimáticas como puedan ser Microsoft Office [48] o Apache OpenOffice [49].

6 BIBLIOGRAFÍA

- [1] «Docker Inc.» [En línea]. Available: <https://www.docker.com/>. [Último acceso: 24 10 2022]
- [2] «Kubernetes» [En línea]. Available: <https://kubernetes.io/docs/concepts/overview/>. [Último acceso: 24 10 2022].
- [3] «PHP - Hypertext Preprocessor» [En línea]. Available: <https://www.php.net/manual/es/intro-what-is.php>. [Último acceso: 01 11 2022].
- [4] «MariaDB - Web» [En línea]. Available: <https://mariadb.org/es/>. [Último acceso: 01 11 2022].
- [5] «Docker y Red Hat» [En línea]. Available: <https://www.redhat.com/es/topics/containers/what-is-docker>. [Último acceso: 13 10 2022].
- [6] G. Wolf, «Contenedores: Un poco de historia» [En línea]. Available: <https://sg.com.mx/revista/52/contenedores-un-poco-historia>. [Último acceso: 13 10 2022]
- [7] «Linux vServer» [En línea]. Available: http://linux-vserver.org/Welcome_to_Linux-VServer.org. [Último acceso: 14 10 2022]
- [8] «Linux Containers» [En línea]. Available: <https://linuxcontainers.org/>. [Último acceso: 21 10 2022].
- [9] «GitHub - LXC» [En línea]. Available: <https://github.com/lxc>. [Último acceso: 21 10 2022].
- [10] «LXC - Introducción» [En línea]. Available: <https://linuxcontainers.org/lxc/introduction/>. [Último acceso: 21 10 2022].
- [11] «Docker Hub». [En línea]. Available: <https://hub.docker.com/>. [Último acceso: 27 10 2022]
- [12] «Docker Desktop». [En línea]. Available: <https://www.docker.com/products/docker-desktop/>. [Último acceso: 27 10 2022]
- [13] «Au-revoir Docker» [En línea]. Available: <https://www.docker.com/blog/au-revoir/>. [Último acceso: 27 10 2022].
- [14] «Microsoft Español». [En línea]. Available: <https://www.microsoft.com/es-es>. [Último acceso: 01 11 2022]
- [15] «CGroups y Namespaces» [En línea]. Available: <https://www.nginx.com/blog/what-are-namespaces-cgroups-how-do-they-work/>. [Último acceso: 04 11 2022].
- [16] «Que es un PID» [En línea]. Available: <https://www.compuhoy.com/que-es-pid-en-linux/>. [Último acceso: 06 11 2022].
- [17] «What's UTS» [En línea]. Available: https://en.wikipedia.org/wiki/Linux_namespaces. [Último acceso: 06 11 2022].

- [18] «Comunicación entre procesos» [En línea]. Available: https://es.wikipedia.org/wiki/Comunicaci%C3%B3n_entre_procesos. [Último acceso: 06 11 2022].
- [19] «Repositorio Docker Hub» [En línea]. Available: <https://hub.docker.com/>. [Último acceso: 08 11 2022].
- [20] «Docker Swarm» [En línea]. Available: <https://docs.docker.com/engine/swarm/>. [Último acceso: 08 11 2022]
- [21] «Docker Blog» [En línea]. Available: <https://www.docker.com/blog/docker-1-12-built-in-orchestration/>. [Último acceso: 09 11 2022].
- [22] «Características Kubernetes» [En línea]. Available: <https://www.redhat.com/es/topics/containers/what-is-kubernetes>. [Último acceso: 10 11 2022].
- [23] «Tipos de Nodos Kubernetes» [En línea]. Available: <https://kubernetes.io/docs/concepts/architecture/nodes/>. [Último acceso: 10 11 2022].
- [24] «Arquitectura Kubernetes» [En línea]. Available: <https://www.redhat.com/es/topics/containers/kubernetes-architecture>. [Último acceso: 11 11 2022].
- [25] «Flannel - Kubernetes» [En línea]. Available: <https://github.com/flannel-io/flannel>. [Último acceso: 11 11 2022].
- [26] «Calico - Kubernetes» [En línea]. Available: <https://projectcalico.docs.tigera.io/getting-started/kubernetes/>. [Último acceso: 12 11 2022].
- [27] RedHat Inc., «Red Hat OpenShift Container Platform» [En línea]. Available: <https://www.redhat.com/cms/managed-files/cl-openshift-4-datasheet-f16726wg-201905-a4-es.pdf>. [Último acceso: 12 11 2022].
- [28] «Kubect!» [En línea]. Available: <https://kubernetes.io/docs/reference/kubect!/>. [Último acceso: 12 11 2022].
- [29] «Nomad - Orchestration made easy» [En línea]. Available: <https://www.nomadproject.io/>. [Último acceso: 13 11 2022].
- [30] «Minikube» [En línea]. Available: <https://minikube.sigs.k8s.io/docs/>. [Último acceso: 13 11 2022].
- [31] «Apache Mesos» [En línea]. Available: <https://mesos.apache.org/>. [Último acceso: 13 11 2022].
- [32] «Kind - Kubernetes» [En línea]. Available: <https://kind.sigs.k8s.io/>. [Último acceso: 15 11 2022].
- [33] «Kubespary - Kubernetes» [En línea]. Available: <https://github.com/kubernetes-sigs/kubespary>. [Último acceso: 15 11 2022].
- [34] «Ansible - Plataforma de configuración de sistemas» [En línea]. Available: [https://es.wikipedia.org/wiki/Ansible_\(software\)](https://es.wikipedia.org/wiki/Ansible_(software)). [Último acceso: 15 11 2022].
- [35] «Node.js» [En línea]. Available: <https://nodejs.org/es/>. [Último acceso: 16 11 2022].

- [36] «React - Biblioteca JavaScript» [En línea]. Available: <https://es.reactjs.org/>. [Último acceso: 16 11 2022].
- [37] «Bootstrap» [En línea]. Available: <https://getbootstrap.com/>. [Último acceso: 18 11 2022].
- [38] «Twitter» [En línea]. Available: <https://twitter.com/?lang=es>. [Último acceso: 18 11 2022].
- [39] «jQuery - Librería JavaScript» [En línea]. Available: <https://jquery.com/>. [Último acceso: 18 11 2022].
- [40] «Nginx» [En línea]. Available: <https://www.nginx.com/>. [Último acceso: 18 11 2022].
- [41] «PHP-FPM» [En línea]. Available: <https://php-fpm.org/>. [Último acceso: 18 11 2022].
- [42] «MariaDB» [En línea]. Available: <https://mariadb.com/>. [Último acceso: 18 11 2022].
- [43] «Flannel para Kubernetes» [En línea]. Available: <https://github.com/flannel-io/flannel>. [Último acceso: 18 11 2022].
- [44] «Amazon Elastic Block Store (EBS)» [En línea]. Available: <https://aws.amazon.com/es/ebs/>. [Último acceso: 22 11 2022].
- [45] «Container Storage Interface (CSI) drivers on Azure Kubernetes Service (AKS)» [En línea]. Available: <https://learn.microsoft.com/en-us/azure/aks/csi-storage-drivers>. [Último acceso: 22 11 2022].
- [46] «Kubernetes cloud provider OpenStack» [En línea]. Available: <https://github.com/kubernetes/cloud-provider-openstack/blob/master/docs/cinder-csi-plugin/using-cinder-csi-plugin.md>. [Último acceso: 22 11 2022].
- [47] «PHPMyAdmin» [En línea]. Available: <https://www.phpmyadmin.net/>. [Último acceso: 24 11 2022].
- [48] «Microsoft Office 365» [En línea]. Available: <https://www.microsoft.com/es-es/microsoft-365>. [Último acceso: 24 11 2022].
- [49] «Apache OpenOffice» [En línea]. Available: <https://www.openoffice.org/es/>. [Último acceso: 27 11 2022].

ANEXO I: CÓDIGO FUENTE DEL PROCEDIMIENTO DE CLASIFICACIÓN DE VEHÍCULOS

```

/*-----*/
/* CLASIFICAMOS los nuevos vehículos de categoría --A--
/* CATEGORIA: A
/*-----*/
function procesa_caracteristicas_vehiculos()
{
    global $db;
    global $session;
    $sql = "SELECT a.numVehiculo as nvehiculo, g.tipo as TIPO, a.idModelo as NumeroModelo, a.matPGC
as PGC, a.matCivil as RESERVADA, a.bastidor as BASTIDOR, a.categoria as CATEGORIA, a.kilometros AS
KILOMETROS, b.nombreMarca as MARCA, c.nombreModelo as MODELO, d.color as COLOR, e.estado as ESTADO,
f.carburante as CARBURANTE ";
    $sql .= "FROM flota.vehiculos as a, flota.marcas as b, flota.modelos as c, flota.colores as d, flota.estados
as e, flota.carburantes as f, flota.tiposVehiculo as g ";
    $sql .= "WHERE a.categoria = 'Z' ";
    $sql .= "AND (a.idMarca = b.idMarca AND a.idModelo = c.idModelo AND a.idColor = d.idColor AND
a.idEstado = e.idEstado AND a.idCarburante = f.idCarburante AND a.idTipoVehiculo = g.idTipo) ";
    $sql .= "ORDER BY a.numVehiculo ASC";
    $result = find_by_sql($sql);

    //$_SESSION['categoria'] = "A";
    $session->msg('s','Listado de vehículos clasificados automáticamente por el sistema.');
```

borra_datos_categoriaTemporal();

```

/*-----*/
/* Declaración de variables
/*-----*/
$num_vehiculo = 0;
$color_v = "";
$tipo_v = "";
$estado_v = "";
$categoria_color = "";
$categoria_tipo = "";
$categoria_estado = "";
$color_cat = "";
$tipo_cat = "";
$estado_cat = "";

/*-----*/
/* Añadimos los datos temporales */
/*-----*/
foreach ($result as $coches)
{
    /*-----*/
    /*-- COLORES ----*/
    /*-----*/
    $num_vehiculo = $coches['nvehiculo'];
    $color_v = $coches['COLOR'];
    $tipo_v = $coches['TIPO'];
    $estado_v = $coches['ESTADO'];

    $categoria_color = busca_categoria_color($color_v);

```

```

if ($categoria_color != null) {
    $color_cat = $categoria_color['categoria'];
} else {
    $color_cat = 'Z';
}
inserta_valor_categoria($num_vehiculo, "Color", $color_cat);
/*-----*/
$categoria_tipo = busca_categoria_tipoVehiculo($tipo_v);
if ($categoria_tipo != null) {
    $tipo_cat = $categoria_tipo['categoria'];
} else {
    $tipo_cat = 'Z';
}
inserta_valor_categoria($num_vehiculo, "Tipo Vehiculo", $tipo_cat);
/*-----*/
$categoria_estado = busca_categoria_estado($estado_v);
if ($categoria_estado != null) {
    $estado_cat = $categoria_estado['categoria'];
} else {
    $estado_cat = 'Z';
}
inserta_valor_categoria($num_vehiculo, "Estado", $estado_cat);
/*-----*/
/* Kilometros
/*-----*/
$kilometros_v = $coches['KILOMETROS'];
$categoria_kms = busca_categoria_baremosClasificacion('kilometros', $kilometros_v);
if ($categoria_kms != null) {
    $kms_cat = $categoria_kms['categoria'];
} else {
    $kms_cat = 'Z';
}
inserta_valor_categoria($num_vehiculo, "Kilometros", $kms_cat);
/*-----*/
/* Cilindrada
/* -
/* Buscamos primero la cilindrada del modelo del vehículo y después
/* averiguamos el baremo de clasificación
/*-----*/

$modelo_a = $coches['NumeroModelo'];
$modelo_v = busca_baremo_modelo('cilindrada', $modelo_a);
if ($modelo_v != null) {
    $modelo_valor = $modelo_v['cilindrada'];
} else {
    $modelo_valor = 0;
}

$categoria_cilin = busca_categoria_baremosClasificacion('cilindrada', $modelo_valor);
if ($categoria_cilin != null) {
    $cilin_cat = $categoria_cilin['categoria'];
} else {
    $cilin_cat = 'Z';
}
inserta_valor_categoria($num_vehiculo, "Cilindrada", $cilin_cat);

/*-----*/
/* CV

```

```

/* -
/* Buscamos primero los CV del modelo del vehículo y después
/* averiguamos el baremo de clasificación
/*-----*/

$cv_a = $coches['NumeroModelo'];
$cv_v = busca_baremo_modelo('CV', $cv_a);
if ($cv_v != null) {
    $cv_valor = $cv_v['CV'];
} else {
    $cv_valor = 0;
}

$categoria_cv = busca_categoria_baremosClasificacion('CV', $cv_valor);
if ($categoria_cv != null) {
    $cv_cat = $categoria_cv['categoria'];
} else {
    $cv_cat = 'Z';
}
inserta_valor_categoria($num_vehiculo, "CV", $cv_cat);

/*-----*/
/* KW
/* -
/* Buscamos primero los KW del modelo del vehículo y después
/* averiguamos el baremo de clasificación
/*-----*/

$kw_a = $coches['NumeroModelo'];
$kw_v = busca_baremo_modelo('KW', $kw_a);
if ($kw_v != null) {
    $kw_valor = $kw_v['KW'];
} else {
    $kw_valor = 0;
}

$categoria_kw = busca_categoria_baremosClasificacion('KW', $kw_valor);
if ($categoria_kw != null) {
    $kw_cat = $categoria_kw['categoria'];
} else {
    $ckw_cat = 'Z';
}
inserta_valor_categoria($num_vehiculo, "KW", $kw_cat);
//-----
}
}

/*-----*/
/* Una vez se han buscado y almacenado los parámetros sobre los baremos
/* pasamos a realizar la clasificación de los vehículos
/*-----*/

function categoriza_vehiculos()
{
    global $db;
    global $session;

```

```

$mensajer = "";

inserta_mensaje_prueba("Estamos empezando...");

$busca = "SELECT numVehiculo as nvehiculo FROM vehiculos WHERE categoria = 'Z' AND idUnidad IS
NULL ORDER BY numVehiculo";
$resultado = find_by_sql($busca);

foreach ($resultado as $coches)
{
    $num_vehiculo = $coches['nvehiculo'];
    $mensajer = "Vehiculo--" . $num_vehiculo;
    inserta_mensaje_prueba("Vehiculo...." . $num_vehiculo);
    $sql      = "SELECT categoria, valor FROM categoriaTemporal WHERE numVehiculo =
'$num_vehiculo'";
    $baremos = find_by_sql($sql);

    $puntosA = 0;
    $puntosB = 0;
    $puntosC = 0;
    $puntosD = 0;
    $categoria_final = "";
    /*-----*/
    /*-- Buscamos la puntuación de los baremos realizados */
    /*-----*/
    foreach ($baremos as $puntos)
    {
        $valorBaremo = $puntos['valor'];
        $categoriaBaremo = $puntos['categoria'];
        inserta_mensaje_prueba("Baremo...Categoria: " . $categoriaBaremo . " - Valor: " .
$valorBaremo);

        switch ($valorBaremo)
        {
            case "A":
                $puntosA++;
                break;
            case "B":
                $puntosB++;
                break;
            case "C":
                $puntosC++;
                break;
            case "D":
                $puntosD++;
                break;
        }
    }
}
}
/*-----*/
/*-- Obtenemos la clasificación mirando los resultados */
/*-----*/
if ($puntosC >= 3) {
    $categoria_final = "C";
}
if ($puntosC >=2 and $puntosB >=3) {
    $categoria_final = "C";
}
if ($puntosB <=2 and $puntosA <=2) {
    $categoria_final = "C";
}

```

```

    }
    if ($puntosA == 3 and $puntosB == 3) {
        $categoria_final = "B";
    }
    if ($puntosA == 4 and $puntosC >=2) {
        $categoria_final = "B";
    }
    if ($puntosB >= 4) {
        $categoria_final = "B";
    }
    if ($puntosA >=6) {
        $categoria_final = "A";
    }
    if ($puntosA == 4 and $puntosB >=2) {
        $categoria_final = "B";
    }
    if ($puntosA == 5 and $puntosB >= 1) {
        $categoria_final = "A";
    }
    if ($categoria_final == "") {
        $categoria_final = "D";
    }
    $mensaje .= " -Puntos A " . $puntosA . " -- Puntos B " . $puntosB . " -- Puntos C " . $puntosC .
" -- Puntuacion Final -> " . $categoria_final;
    inserta_mensaje_prueba($mensaje);
    $_SESSION["mensaje"] = $mensaje;
    actualiza_categoria($categoria_final, $num_vehiculo);
} --- Del Foreach
$session->msg('d','Vehículos clasificados automáticamente por el sistema.');
```

ANEXO II: CÓDIGO FUENTE DEL PROCEDIMIENTO DE ADJUDICACIÓN DE VEHÍCULOS

```

/*-----*/
/*-- (1) Buscamos los vehículos sin adjudicar por categoría */
/*-----*/
function busca_vehiculos_sin_adjudicar($categoria)
{
    global $db;
    $p_cat = remove_junk($db->escape($categoria));
    $sql = $db->query("SELECT idVehiculo, numVehiculo, categoria FROM vehiculos WHERE (categoria
= '$p_cat') AND (adjudicado = 0 or idUnidad IS NULL) AND (adjudicable = 1)");
    if($result = $db->fetch_assoc($sql))
        return $result;
    else
        return null;
} /*----- Fin Procedimiento ----- */

/*-----*/
/*-- (2) Buscamos las Unidades a las que hacer el reparto */
/*-----*/
function busca_unidades_reparto()
{
    global $db;
    $busca = "SELECT idUnidad, departamento, grupo FROM unidades WHERE activarReparto = 1";
    $resultado = find_by_sql($busca);
    return $resultado;
} /*----- Fin Procedimiento ----- */

/*-----*/
/*-- (3) Buscamos las puntuaciones para el reparto de una categoría */
/*-----*/
function busca_puntuaciones_reparto($categoria)
{
    global $db;
    $busca = "";
    $p_cat = remove_junk($db->escape($categoria));
    $busca = "SELECT categoria, jefeArea, operativo, obtencion, analista, administrativo FROM
puntuacionReparto WHERE categoria = '$p_cat' ";
    $resultado = find_by_sql($busca);
    return $resultado;
} /*----- Fin Procedimiento ----- */

/*-----*/
/*-- (4) Buscamos el catálogo que tiene una Unidad */
/*-----*/
function busca_catalogo_unidad($unidad)
{
    global $db;
    $p_unidad = remove_junk($db->escape($unidad));
    $busca = "SELECT idCatalogo, jefeArea, operativo, obtencion, analista, administrativo FROM catalogos
WHERE idUnidad = '$p_unidad' ";
    $resultado = find_by_sql($busca);
    return $resultado;
} /*----- Fin Procedimiento ----- */

/*-----*/

```

```

/*-- (5) Buscamos las vacantes del catálogo que tiene una Unidad */
/*-----*/
function busca_vacantes_catologo_unidad($unidad)
{
    global $db;
    $p_unidad = remove_junk($db->escape($unidad));
    $busca = "SELECT idCatalogoVacante, jefeArea, operativo, obtencion, analista, administrativo FROM
catalogosVacante WHERE idUnidad = '$p_unidad' ";
    $resultado = find_by_sql($busca);
    return $resultado;
} /*----- Fin Procedimiento ----- */

/*-----*/
/*-- (6) Borramos los datos de las tablas */
/*-----*/
function borra_datos_tablas_asignacion()
{
    global $db;
    $sql = $db->query("DELETE FROM mensajePrueba");
    $sql = $db->query("DELETE FROM puntuacionUnidades");
    $sql = $db->query("DELETE FROM puntuacionVehiculos");
    $sql = $db->query("DELETE FROM puntuacionFinal");
}

/*-----*/
/*-- (-6Bis) Borramos los datos de las tablas */
/*-- de puntuación Vehiculos y Final cada vez que */
/*-- pasamos de Categoría */
/*-----*/
function borra_datos_tablas_manioobra()
{
    global $db;
    $sql = $db->query("DELETE FROM puntuacionVehiculos");
    $sql = $db->query("DELETE FROM puntuacionFinal");
}

/*-----*/
/*-- (7) Añadimos, en la tabla puntuacionUnidades */
/*-- los datos que hemos calculado para su posterior utilización --*/
/*-----*/
function inserta_datos_reparto($idUnidad, $categoria, $suma, $porcentaje)
{
    global $db;
    $sql = "INSERT INTO puntuacionUnidades (idUnidad, categoria, suma, porcentaje) VALUES ( ";
    $sql .= "'{$idUnidad}', '{$categoria}', '{$suma}', '{$porcentaje}')";
    $result = $db->query($sql);
    return ($result && $db->affected_rows() === 1 ? true : false);
}

/*-----*/
/*-- (8) Buscamos los vehículos que tiene adjudicados una unidad */
/*-- en una categoría */
/*-----*/
function busca_vehiculos_unidad($idUnidad, $categoria)
{
    global $db;
    $sql = "SELECT COUNT(idVehiculo) AS total FROM vehiculos WHERE idUnidad ='$idUnidad' AND
categoria = '$categoria'";

```

```

        $result = $db->query($sql);
        return($db->fetch_assoc($result));
    }

    /*-----*/
    /*-- (9) Añadimos, en la tabla puntuacionVehiculos */
    /*-- los datos que hemos calculado para su posterior utilización --*/
    /*-----*/
    function inserta_datos_reparto_vehiculos($idUnidad, $categoria, $suma, $porcentaje)
    {
        global $db;
        $sql = "INSERT INTO puntuacionVehiculos (idUnidad, categoria, suma, porcentaje) VALUES ( ";
        $sql .= "'{$idUnidad}', '{$categoria}', '{$suma}', '{$porcentaje}');"
        $result = $db->query($sql);
        return ($result && $db->affected_rows() === 1 ? true : false);
    }

    /*-----*/
    /*-- (10) Buscamos las Puntuaciones de la Unidad del Catálogo */
    /*-----*/
    function busca_puntuaciones_catalogo_unidad($idUnidad, $categoria)
    {
        global $db;
        $sql = $db->query("SELECT suma, porcentaje FROM puntuacionUnidades WHERE idUnidad
        ='$idUnidad' AND categoria = '$categoria' LIMIT 1");
        if($result = $db->fetch_assoc($sql))
            return $result;
        else
            return null;
    } /*----- Fin Procedimiento ----- */

    /*-----*/
    /*-- (11) Buscamos las Puntuaciones de la Unidad de los vehiculos que YA TIENE ADJUDICADOS */
    /*-----*/
    function busca_puntuaciones_vehiculos_unidad($idUnidad, $categoria)
    {
        global $db;
        $sql = "SELECT suma, porcentaje FROM puntuacionVehiculos WHERE idUnidad ='$idUnidad' AND
        categoria = '$categoria'";
        $result = $db->query($sql);
        return($db->fetch_assoc($result));
    } /*----- Fin Procedimiento ----- */

    /*-----*/
    /*-- (13) Añadimos, en la tabla puntuacionFinal */
    /*-- los datos que hemos calculado para realizar el reparto final --*/
    /*-----*/
    function inserta_datos_puntuacion_final($idUnidad, $categoria, $suma, $porcentaje)
    {
        global $db;
        $sql = "INSERT INTO puntuacionFinal (idUnidad, categoria, suma, porcentaje) VALUES ( ";
        $sql .= "'{$idUnidad}', '{$categoria}', '{$suma}', '{$porcentaje}');"
        $result = $db->query($sql);
        return ($result && $db->affected_rows() === 1 ? true : false);
    }

    /*-----*/
    /*-- (14) Adjudicamos, un vehículo a una Unidad */

```

```

/*-- Es la PRE-adjudicación. Faltaría confirmarlo */
/*-----*/
function adjudica_vehiculo_unidad($num_vehiculo, $idUnidad)
{
    global $db;
    $sql = "UPDATE vehiculos SET idUnidad = '{$idUnidad}' WHERE idVehiculo = '{$num_vehiculo}'";
    $result = $db->query($sql);
    return($db->affected_rows() === 1 ? true : false);
}
/*-----*/
/*-- (15) Busca la Unidad a la que hay que adjudicar el vehículo */
/*-----*/
function busca_primera_unidad_adjudicar($categoria)
{
    global $db;
    $sql = $db->query("SELECT idUnidad, porcentaje FROM puntuacionFinal WHERE categoria =
'$categoria' ORDER BY porcentaje DESC LIMIT 1");
    if($result = $db->fetch_assoc($sql))
        return $result;
    else
        return null;
} /*----- Fin Procedimiento ----- */

/*-----*/
/*-- (16) Validamos definitivamente los vehiculos asignados a un Grupo */
/*-----*/
function validar_vehiculos_adjudicados()
{
    global $db;
    $sql = "UPDATE vehiculos SET adjudicado = 1 WHERE idUnidad > 0 and adjudicado = 0";
    $result = $db->query($sql);
    return($db->affected_rows() === 1 ? true : false);
}

/*-----*/
/*-- (PROCEDIMIENTO PRINCIPAL) */
/*-- Realizamos todos los procesos para la adjudicación */
/*-- de los nuevos vehículos */
/*-----*/
function adjudica_nuevos_vehiculos()
{
    $categorias = array("A","B","C");
    $longitud = count($categorias);
    borra_datos_tablas_asignacion();

    /*-----*/
    /*-- Recorremos las tres categorías --*/
    /*-----*/
    for ($i=0; $i<$longitud; $i++)
    {
        $categoria = $categorias[$i];
        $coches = busca_vehiculos_sin_adjudicar($categoria);
        $unidades = busca_unidades_reparto();
        $puntuaciones = busca_puntuaciones_reparto($categoria);

        //$puntua_unidad [][];
        //$vacante_unidad [][];
    }
}

```

```

//$puestos_actuales_unidad = [[]];
$puntuacion_unidad = 0;
$total_puntuacion_unidad = 0; /*--- de un Departamento ---*/
$total_puntuacion_total = 0; /*--- de TODA la Unidad ---*/
$total_componentes = 0; /*-- De un Departamento --*/
$total_componentes_unidad = 0; /*-- De TODA la Unidad ---*/
$indice1 = 0;
$indice2 = 0;
    $paso = 0;

foreach($unidades as $unidad)
{
    /*-----*/
    /*- Averiguamos el catálogo */
    /*-----*/
    $idUni = $unidad['idUnidad'];
    $catalogo_unidad = busca_catalogo_unidad($idUni);
    $indice1 = 0;
    $paso++;

    $nombre_Unidad = $unidad['departamento'] . "--" . $unidad['grupo'];
    inserta_mensaje_prueba("Unidad: " . $nombre_Unidad);

    foreach ($catalogo_unidad as $cat_unidad)
    {
/*-- Almacenamos en el Array los datos de idUnidad - CATALOGO DE PUESTOS DE TRABAJO -- */
        $indice1++;
        $puntua_unidad [$idUni][$indice1] = $cat_unidad['jefeArea'];
        //inserta_mensaje_prueba("Indices JefeArea...: " . $idUni . "--" . $indice1);
        $indice1++;
        $puntua_unidad [$idUni][$indice1] = $cat_unidad['operativo'];
        //inserta_mensaje_prueba("Indices Operativo..: " . $idUni . "--" . $indice1);
        $indice1++;
        $puntua_unidad [$idUni][$indice1] = $cat_unidad['obtencion'];
        //inserta_mensaje_prueba("Indices Obtencion..: " . $idUni . "--" . $indice1);
        $indice1++;
        $puntua_unidad [$idUni][$indice1] = $cat_unidad['analista'];
        //inserta_mensaje_prueba("Indices Analista...: " . $idUni . "--" . $indice1);
        $indice1++;
        $puntua_unidad [$idUni][$indice1] = $cat_unidad['administrativo'];
        //inserta_mensaje_prueba("Indices Administrativo..: " . $idUni . "--" . $indice1);
    } /*-- Del FOR de Catalogo_Unidad ----*/

    /*-----*/
    /*-- Obtenemos las puntuaciones */
    /*-----*/
    $indice1 = 0;
    foreach ($puntuaciones as $puntos)
    {
        $indice1++;
        $puntos_reparto [1][$indice1] = $puntos['jefeArea'];

        $indice1++;

        $puntos_reparto [1][$indice1] = $puntos['operativo'];
        $indice1++;
        $puntos_reparto [1][$indice1] = $puntos['obtencion'];
    }
}

```

```

        $indice1++;
        $puntos_reparto [1][$indice1] = $puntos['analista'];

        $indice1++;
        $puntos_reparto [1][$indice1] = $puntos['administrativo'];

    } /*--- Del FOR de Puntuaciones ---*/

    $vacantes_unidad = busca_vacantes_catalogo_unidad($idUni);
    $indice1 = 1;
    $total_puntuacion_unidad = 0;

    /*-----*/
    /*-- En este FOR realizamos las siguientes acciones */
    /*-- */
    /*-- Buscamos las VACANTES de la Unidad */
    /*-- Restamos, del CATALOGO, las VACANTES = puestos_actuales_unidad */
    /*-- Sumamos el TOTAL DE COMPONENTES de toda la Unidad */
    /*-- Calculamos la PUNTUACIÓN de la Unidad */
    /*-- Averiguamos el TOTAL DE PUNTUACIÓN de la Unidad */
    /*-----*/

    foreach ($vacantes_unidad as $vac_unidad)
    {
        $indice2 = 1;
        $puntuacion = 0;
        $componentes = 0;
        /*-- Almacenamos en el Array los datos de idUnidad - VACANTES DE PUESTOS
DE TRABAJO -- */
        $vacante_unidad [$idUni][$indice1] = $vac_unidad['jefeArea'];
        $componentes = $puntuacion_unidad [$idUni][$indice1] - $vacante_unidad
[$idUni][$indice1];

        $total_componentes = $total_componentes + $componentes;
        $puntuacion = $componentes * $puntos_reparto [1][$indice2];
        $total_puntuacion_unidad = $total_puntuacion_unidad + $puntuacion;
        //inserta_mensaje_prueba("Jefe Area.: Catalogo: " . $puntuacion_unidad
[$idUni][$indice1] . " -- Vacantes: " . $vacante_unidad [$idUni][$indice1] . "-- Final: " . $componentes);
        //inserta_mensaje_prueba("Puntuacion reparto Jefe Area.: " . $puntuacion);
        $indice1++;
        $indice2++;

        $puntuacion = 0;
        $componentes = 0;
        $vacante_unidad [$idUni][$indice1] = $vac_unidad['operativo'];
        $componentes = $puntuacion_unidad [$idUni][$indice1] - $vacante_unidad
[$idUni][$indice1];

        $total_componentes = $total_componentes + $componentes;
        $puntuacion = $componentes * $puntos_reparto [1][$indice2];
        $total_puntuacion_unidad = $total_puntuacion_unidad + $puntuacion;
        //inserta_mensaje_prueba("Operativo.: Catalogo: " . $puntuacion_unidad
[$idUni][$indice1] . " -- Vacantes: " . $vacante_unidad [$idUni][$indice1] . "-- Final: " . $componentes );
        //inserta_mensaje_prueba("Puntuacion reparto Operativo.: " . $puntuacion);
        $indice1++;
        $indice2++;

        $puntuacion = 0;
        $componentes = 0;
        $vacante_unidad [$idUni][$indice1] = $vac_unidad['obtencion'];

```

```

$componentes = $puntua_unidad [$idUni][$indice1] - $vacante_unidad
[$idUni][$indice1];
$total_componentes = $total_componentes + $componentes;
$puntuacion = $componentes * $puntos_reparto [1][$indice2];
$total_puntuacion_unidad = $total_puntuacion_unidad + $puntuacion;
//inserta_mensaje_prueba("Obtencion.: Catalogo: " . $puntua_unidad
[$idUni][$indice1] . " -- Vacantes: " . $vacante_unidad [$idUni][$indice1] . "-- Final: " . $componentes);
//inserta_mensaje_prueba("Puntuacion reparto Obtencion..: " . $puntuacion);
$indice1++;
$indice2++;

$puntuacion = 0;
$componentes = 0;
$vacante_unidad [$idUni][$indice1] = $vac_unidad['analista'];
$componentes = $puntua_unidad [$idUni][$indice1] - $vacante_unidad
[$idUni][$indice1];
$total_componentes = $total_componentes + $componentes;
$puntuacion = $componentes * $puntos_reparto [1][$indice2];
$total_puntuacion_unidad = $total_puntuacion_unidad + $puntuacion;
//inserta_mensaje_prueba("Analista.: Catalogo: " . $puntua_unidad
[$idUni][$indice1] . " -- Vacantes: " . $vacante_unidad [$idUni][$indice1] . "-- Final: " . $componentes );
//inserta_mensaje_prueba("Puntuacion reparto Analista..: " . $puntuacion);
$indice1++;
$indice2++;

$puntuacion = 0;
$componentes = 0;
$vacante_unidad [$idUni][$indice1] = $vac_unidad['administrativo'];
$componentes = $puntua_unidad [$idUni][$indice1] - $vacante_unidad
[$idUni][$indice1];
$total_componentes = $total_componentes + $componentes;
$puntuacion = $componentes * $puntos_reparto [1][$indice2];
$total_puntuacion_unidad = $total_puntuacion_unidad + $puntuacion;
//inserta_mensaje_prueba("Administrativo.: Catalogo: " . $puntua_unidad
[$idUni][$indice1] . " -- Vacantes: " . $vacante_unidad [$idUni][$indice1] . "-- Final: " . $componentes );
//inserta_mensaje_prueba("Puntuacion reparto Administrativo..: " .
$puntuacion);

//inserta_mensaje_prueba("Total Componentes Departamento.: " .
$total_componentes);
//inserta_mensaje_prueba("Total Puntuacion Departamento.: " .
$total_puntuacion_unidad);
//inserta_mensaje_prueba("-----");
//inserta_mensaje_prueba("-----");
$puntuacion_grupo[$idUni][$paso] = $total_puntuacion_unidad;

} /*-- Del FOR de las vacantes de la_Unidad ----*/

/*----- Son TODOS los componentes de la UNIDAD -----*/
$total_componentes_unidad = $total_componentes_unidad + $total_componentes;

/*----- Es la puntuación TOTAL de TODA la Unidad de esta categoría -*/
$total_puntuacion_total = $total_puntuacion_total + $total_puntuacion_unidad;

$total_componentes = 0;
$total_puntuacion_unidad = 0;

```

```

    } /*-- Del FOR de Unidades -----*/

    inserta_mensaje_prueba("Total Componentes Unidad.: " . $total_componentes_unidad);
    inserta_mensaje_prueba("Total Puntuacion Unidad.: " . $total_puntuacion_total);

    /*-----*/
    /*-- Una vez tenemos los datos de todos los Grupos, calculamos el porcentaje de cada uno de ellos */
    /*-----*/

    $paso = 0;
    foreach ($unidades as $unidad)
    {
        $paso++;
        $idUni = $unidad['idUnidad'];
        $valor_total = $total_puntuacion_total;
        $puntos_grupo = $puntuacion_grupo[$idUni][$paso];
        if ($puntos_grupo != null) {
            $calculo_grupo = ($puntos_grupo * 100) / $valor_total;
        } else {
            $calculo_grupo = 0;
        }
        //inserta_mensaje_prueba("Unidad.: " . $idUni . "--" . $puntos_grupo . "--" .
$calculo_grupo);
        inserta_datos_reparto($idUni, $categorias[$i], $puntos_grupo, $calculo_grupo);
    } /*--- Del FOR de Unidades para calcular el porcentaje ----*/

    /*-----*/
    /*-- A partir de aquí es cuando tenemos que repartir los vehículos de */
    /*-- la categoría actual, porque a partir de aquí es cuando se vuelven */
    /*-- a calcular los porcentajes de vehículos, dado que ya tenemos los */
    /*-- calculos de catalogo, plantilla y vacantes */
    /*-----*/

    $cacharros = busca_vehiculos_general($categorias[$i]);
    $cuantos = count($cacharros);
    inserta_mensaje_prueba("Categoría.: " . $categorias[$i] . " .. Cuantos --" . $cuantos);

    foreach ($cacharros as $coche)
    {
        borra_datos_tablas_maniobra();
        inserta_mensaje_prueba("Entrando...");

        $nnncoche = $coche['numVehiculo'];

        /*-----*/
        /*-- Buscamos los vehículos que tiene cada Unidad adjudicados de cada categoría ---*/
        /*-----*/
        $suma_Vehiculos = 0;
        $paso = 0;
        foreach ($unidades as $unidad)
        {
            $paso++;
            $idUni = $unidad['idUnidad'];
            $nVehiculos = busca_vehiculos_unidad($idUni, $categorias[$i]);
            if ($nVehiculos != null) {
                $numTotal = $nVehiculos['total'];
            }
        }
    }

```

```

    } else {
        $numTotal = 0;
    }
    $vehiculosUnidad[$idUni][$paso] = $numTotal;
    $suma_Vehiculos = $suma_Vehiculos + $numTotal;

    inserta_mensaje_prueba("Coches de la Unidad..." . $numTotal . "--" . $idUni . "-
-Paso:" . $paso);
} /*--- Del FOR de Unidades para averiguar los vehículos que tiene adjudicados ----*/

/*-----*/
/*-- Una vez tenemos los datos de todos los Vehículos que tienen las Unidades, calculamos */
/*-- el porcentaje de cada una de ellas */
/*-----*/
    $paso = 0;
    foreach ($unidades as $unidad)
    {
        $paso++;
        $idUni = $unidad['idUnidad'];
        $valor_total = $suma_Vehiculos;
        $vehiculos_grupo = $vehiculosUnidad[$idUni][$paso];
        if ($vehiculos_grupo > 0) {
            $calculo_grupo = ($vehiculos_grupo * 100) / $suma_Vehiculos;
        } else {
            $calculo_grupo = 0;
        }

        inserta_datos_reparto_vehiculos($idUni,
$categorias[$i],$vehiculos_grupo,$calculo_grupo);
    } /*--- Del FOR de Unidades para calcular el porcentaje ----*/

/*-----*/
/*-- Calculamos el PORCENTAJE FINAL DE REPARTO */
/*-- y los almacenamos en puntuacionFinal */
/*-----*/

    $paso = 0;
    foreach ($unidades as $unidad)
    {
        $paso++;
        $idUni = $unidad['idUnidad'];
        $categoria = $categorias[$i];

        $puntosvehiculo = busca_puntuaciones_vehiculos_unidad($idUni, $categoria);
        $puntosUnidad = busca_puntuaciones_catalogo_unidad($idUni, $categoria);

        if ($puntosvehiculo != null) {
            $vehiculo_suma = $puntosvehiculo['suma'];
            $vehiculo_porcentaje = $puntosvehiculo['porcentaje'];
        } else {
            $vehiculo_suma = 0;
            $vehiculo_porcentaje = 0;
        }

        if ($puntosUnidad != null) {
            $unidad_suma = $puntosUnidad['suma'];

```

```

        $unidad_porcentaje = $puntosUnidad['porcentaje'];
    } else {
        $unidad_suma = 0;
        $unidad_porcentaje = 0;
    }
    $suma_final = $unidad_suma - $vehiculo_suma;
    $porcentaje_final = $unidad_porcentaje - $vehiculo_porcentaje;

    inserta_datos_puntuacion_final($idUni, $categoria, $suma_final, $porcentaje_final);

    //inserta_mensaje_prueba("Unidad.." . $idUni . "----Categoria: " . $categoria);
    //inserta_mensaje_prueba("Cuenta.." . $paso . "-- Suma Unidad: " . $unidad_suma . "-
- Porcentaje Unidad .: " . $unidad_porcentaje);
    //inserta_mensaje_prueba("Cuenta.." . $paso . "-- Suma Vehiculo: " . $vehiculo_suma
. "-- Porcentaje Vehiculo .: " . $vehiculo_porcentaje);
} /*----- Del FOR de los Puntos de la Unidad -----*/

/*-----*/
/*-- Seleccionamos el primer vehículo */
/*-- que tenemos que adjudicar de la */
/*-- CATEGORIA ACTUAL */
/*-----*/

$numeroCoche = $coche['idVehiculo'];
$buscaCandidatos = busca_primera_unidad_adjudicar($categoria);
if ($buscaCandidatos != null) {
    $unidad_adjudicataria = $buscaCandidatos['idUnidad'];
    $adjudicacion = adjudica_vehiculo_unidad($numeroCoche,
$unidad_adjudicataria);
    inserta_mensaje_prueba("Adjudicacion Vehiculo.." . $numeroCoche . " -- A la
Unidad.." . $unidad_adjudicataria);
} else {
    $unidad_adjudicataria = null;

    inserta_mensaje_prueba("--ERROR-- NO se ha encontrado una Unidad para
adjudicar el coche");
}
//if ($categorias[$i] == "A" OR $categorias[$i] == "B")
//{
//    borra_datos_tablas_manobra();
//}
} /*----- Del FOR de los COCHES DE LA CATEGORIA -----*/

} /*-- Del FOR de las tres categorías ----*/

} /*----- Fin Procedimiento ----- */

```