|  |
| --- |
| Diseño de Microservicios  Bajo Python |
| I.E.S. Arcipreste de Hita  César San Juan Pastor  Dpto. de Informática  marzo 2022 – junio 2022 |



Contenido

[Capítulo I. Introducción 6](#_Toc104188409)

[Antecedentes 6](#_Toc104188410)

[Qué es un microservicio 6](#_Toc104188411)

[Conceptos claves de los ms 12](#_Toc104188412)

[Comparación de monolítico vs microservicios 12](#_Toc104188413)

[Ventajas de los desarrollos monolíticos 12](#_Toc104188414)

[Desventajas de los desarrollos monolíticos 12](#_Toc104188415)

[Ventajas de los desarrollos con microservicios 12](#_Toc104188416)

[Desventajas de los desarrollos con microservicios 13](#_Toc104188417)

[Cómo modelar los microservicios 13](#_Toc104188418)

[Tipos de acoplamiento 14](#_Toc104188419)

[Metodologías a usar 15](#_Toc104188420)

[Capítulo II. Conceptos de desarrollo 16](#_Toc104188421)

[Partiendo de un desarrollo anterior 16](#_Toc104188422)

[Tener un objetivo claro 16](#_Toc104188423)

[Migración incremental 16](#_Toc104188424)

[Estudio detallado del dominio del problema 16](#_Toc104188425)

[Descomposición por capas 17](#_Toc104188426)

[Patrones de diseño para la descomposición 17](#_Toc104188427)

[Consideraciones sobre los datos 18](#_Toc104188428)

[Comunicaciones entre los ms 18](#_Toc104188429)

[Tecnologías de comunicación 19](#_Toc104188430)

[Implementación de la comunicación 20](#_Toc104188431)

[Opciones tecnológicas 20](#_Toc104188432)

[Formatos de la serialización 20](#_Toc104188433)

[Esquemas del API 20](#_Toc104188434)

[Manejo de los cambios en los ms 21](#_Toc104188435)

[Principio DRY (Don’t Repeat Yourself) 21](#_Toc104188436)

[Descubrimiento de servicios 21](#_Toc104188437)

[Gestión del crecimiento 21](#_Toc104188438)

[Servicio de documentación 22](#_Toc104188439)

[Capítulo III. Implementación 23](#_Toc104188440)

[Transacciones 23](#_Toc104188441)

[Gestión del código 23](#_Toc104188442)

[Desarrollo del código 23](#_Toc104188443)

[Organización del código 24](#_Toc104188444)

[Propiedad del código 24](#_Toc104188445)

[Despliegue 24](#_Toc104188446)

[Test de la aplicación 26](#_Toc104188447)

[Implementación del test de aceptación 26](#_Toc104188448)

[Implementación de otros tipos de test 27](#_Toc104188449)

[Monitorización 27](#_Toc104188450)

[Seguridad 28](#_Toc104188451)

[Escalado 29](#_Toc104188452)

[Capítulo IV. Uso de Docker 31](#_Toc104188453)

[Arquitectura 31](#_Toc104188454)

[Objetos 32](#_Toc104188455)

[Instalación de Docker 33](#_Toc104188456)

[Imágenes Docker 35](#_Toc104188457)

[El archivo Dockerfile 37](#_Toc104188458)

[Creación de imágenes 41](#_Toc104188459)

[Capas de una imagen 42](#_Toc104188460)

[Estructura de las órdenes Docker 46](#_Toc104188461)

[Gestión de las imágenes: docker image 46](#_Toc104188462)

[Contenedores 47](#_Toc104188463)

[Ciclo de vida de un contenedor 47](#_Toc104188464)

[Gestión de los contenedores: docker container 48](#_Toc104188465)

[Creación y ejecución de contenedores: docker container run 48](#_Toc104188466)

[Listado de contenedores en ejecución: docker container ls 50](#_Toc104188467)

[Usos comunes con los contenedores 51](#_Toc104188468)

[Gestión de los puertos de comunicación 52](#_Toc104188469)

[Políticas de reinicio de contenedores 53](#_Toc104188470)

[Almacenamiento 54](#_Toc104188471)

[Bind mounts 54](#_Toc104188472)

[Volúmenes 55](#_Toc104188473)

[Órdenes de gestión 56](#_Toc104188474)

[Resúmenes de comandos sintaxis antigua 56](#_Toc104188475)

[Comandos interesantes 57](#_Toc104188476)

[Redes bajo Docker 57](#_Toc104188477)

[Arquitectura 57](#_Toc104188478)

[Drivers de red 57](#_Toc104188479)

[Gestión de las redes: docker network 58](#_Toc104188480)

[Creación de redes: docker network create 59](#_Toc104188481)

[Conectar un contenedor a una red: docker network connect 60](#_Toc104188482)

[Ejemplo de uso: Mysql – WordPress – PhpMyAdmin 61](#_Toc104188483)

[Docker compose 63](#_Toc104188484)

[El archivo de configuración 63](#_Toc104188485)

[Gestión de docker compose: docker-compose 64](#_Toc104188486)

[Ejemplo de aplicación: Mysql – WordPress – PhpMyAdmin 66](#_Toc104188487)

[Capítulo V. Análisis y diseño de los ms 68](#_Toc104188488)

[Análisis del sistema monolítico 68](#_Toc104188489)

[Análisis de páginas y datos 68](#_Toc104188490)

[Análisis del funcionamiento: login - logout 69](#_Toc104188491)

[Análisis del funcionamiento: Registro de entrada 70](#_Toc104188492)

[Análisis del funcionamiento: Registro de salida 70](#_Toc104188493)

[Análisis del funcionamiento: Exportar 71](#_Toc104188494)

[Análisis del funcionamiento: Listados registro entrada 71](#_Toc104188495)

[Análisis del funcionamiento: Listados registro salida 72](#_Toc104188496)

[Diseño del sistema basado en ms 72](#_Toc104188497)

[Desarrollo temporal 73](#_Toc104188498)

[Arquitecturas elegidas 74](#_Toc104188499)

[Diseño ms autorización 75](#_Toc104188500)

[Diseño del API 75](#_Toc104188501)

[Diseño ms rentrada 76](#_Toc104188502)

[Diseño del API 76](#_Toc104188503)

[Diseño ms rsalida 77](#_Toc104188504)

[Diseño del API 77](#_Toc104188505)

[Diseño ms ui usuario 77](#_Toc104188506)

[Diseño del API 77](#_Toc104188507)

[Diseño ms ui administrador 78](#_Toc104188508)

[Capítulo VI Implementación 79](#_Toc104188509)

[Implementación de la BBDD 79](#_Toc104188510)

[Scripts de creación 80](#_Toc104188511)

[Servicio de autorización 85](#_Toc104188512)

[wsgi.py 86](#_Toc104188513)

[Directorio auth 86](#_Toc104188514)

[Dockerizar el servicio 96](#_Toc104188515)

[Servicio rentrada 100](#_Toc104188516)

[Directorio auth 100](#_Toc104188517)

[Dockerizar el servicio 108](#_Toc104188518)

[Servicio rsalida 109](#_Toc104188519)

[Servicio ui usuario 109](#_Toc104188520)

[Directorio ui 109](#_Toc104188521)

[Servicio ui admin 117](#_Toc104188522)

[Ejecución con docker-compose 117](#_Toc104188523)

[docker-compose.yaml 117](#_Toc104188524)

# Capítulo I. Introducción

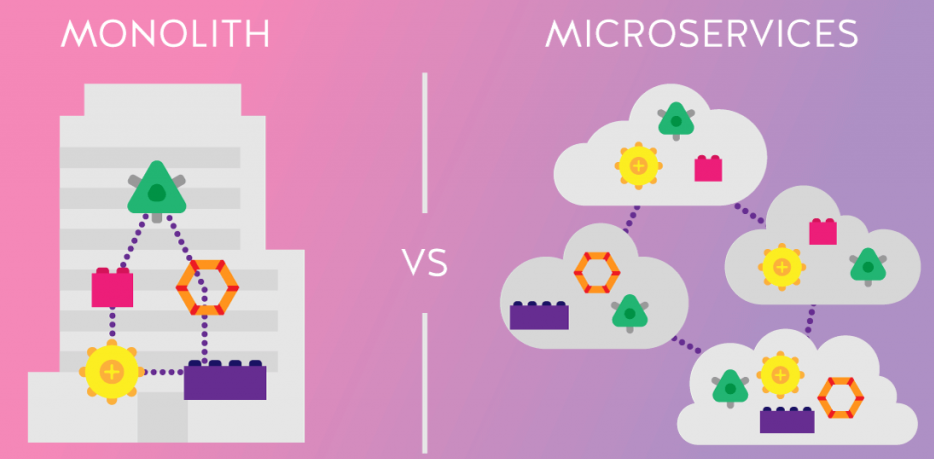
## Antecedentes

Tradicionalmente el desarrollo web se ha realizado mediante aplicaciones monolíticas creadas por uno o varios equipos de programadores. Esta estructura no implica que el software no tenga una estructura interna, de hecho, la mayoría de los proyectos se basan en el modelo MVC, un patrón muy estudiado y extendido en el desarrollo de aplicaciones Web.

Este tipo de desarrollos han presentado un problema con la explosión de la red dada en estos últimos años. En concreto, una aplicación monolítica solo puede crecer usando más recursos para servir más peticiones, es el único modelo válido. Si necesito servir más peticiones necesitaré más servidores, o más memoria, o duplicaré el servicio, pero no permite otro tipo de escalado.

Además, el código que genera, aunque tendrá una estructura, crecerá de forma desmesurada y su gestión y actualización será cada vez más difícil. Los modelos actuales de servicios (Netflix, Azure, AWS, etc.) demandan otro tipo muy diferente de programación, que permita gestionar mejor el código y sus cambios, así como una mejora en la capacidad de los mismos (escalado) de forma sencilla.

Si en vez de tener una aplicación que lo hace todo, tenemos múltiples aplicaciones pequeñas que cooperan entre ellas para realizar el trabajo, tendremos un nuevo modelo. Si además ese modelo reside en diferentes procesos (y/o servidores), pasamos de un modelo centralizado a un modelo distribuido de ejecución.



## Qué es un microservicio

Un microservicio es un servicio independiente que modela un dominio de negocio. Encapsula toda la funcionalidad de un dominio de negocio en un único lugar y es accesible vía red a otros microservicios.

En estos apuntes utilizaremos una aplicación monolítica y la transformaremos una aplicación basada en microservicios.

Tenemos creada una aplicación para el registro de entrada y salida de documentación de nuestro instituto. Esta aplicación permite crear entradas en el registro de entrada y salida, así como los listados de ambos. También permite exportar todo el registro como copia de seguridad en formato Excel (csv). Toda la aplicación está protegida y solo se permite el acceso mediante clave a la misma.

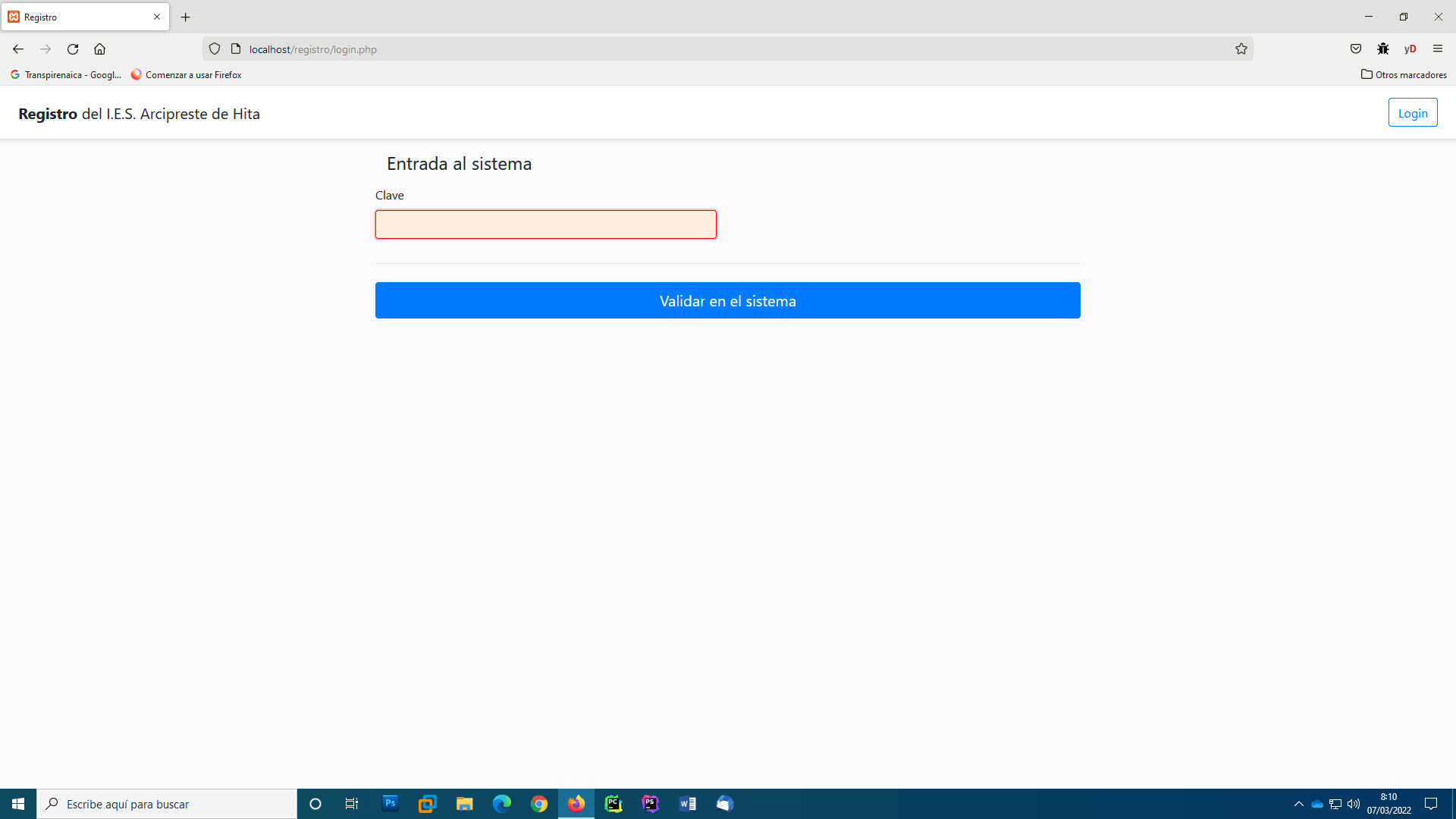


Ilustración Ventana de Login

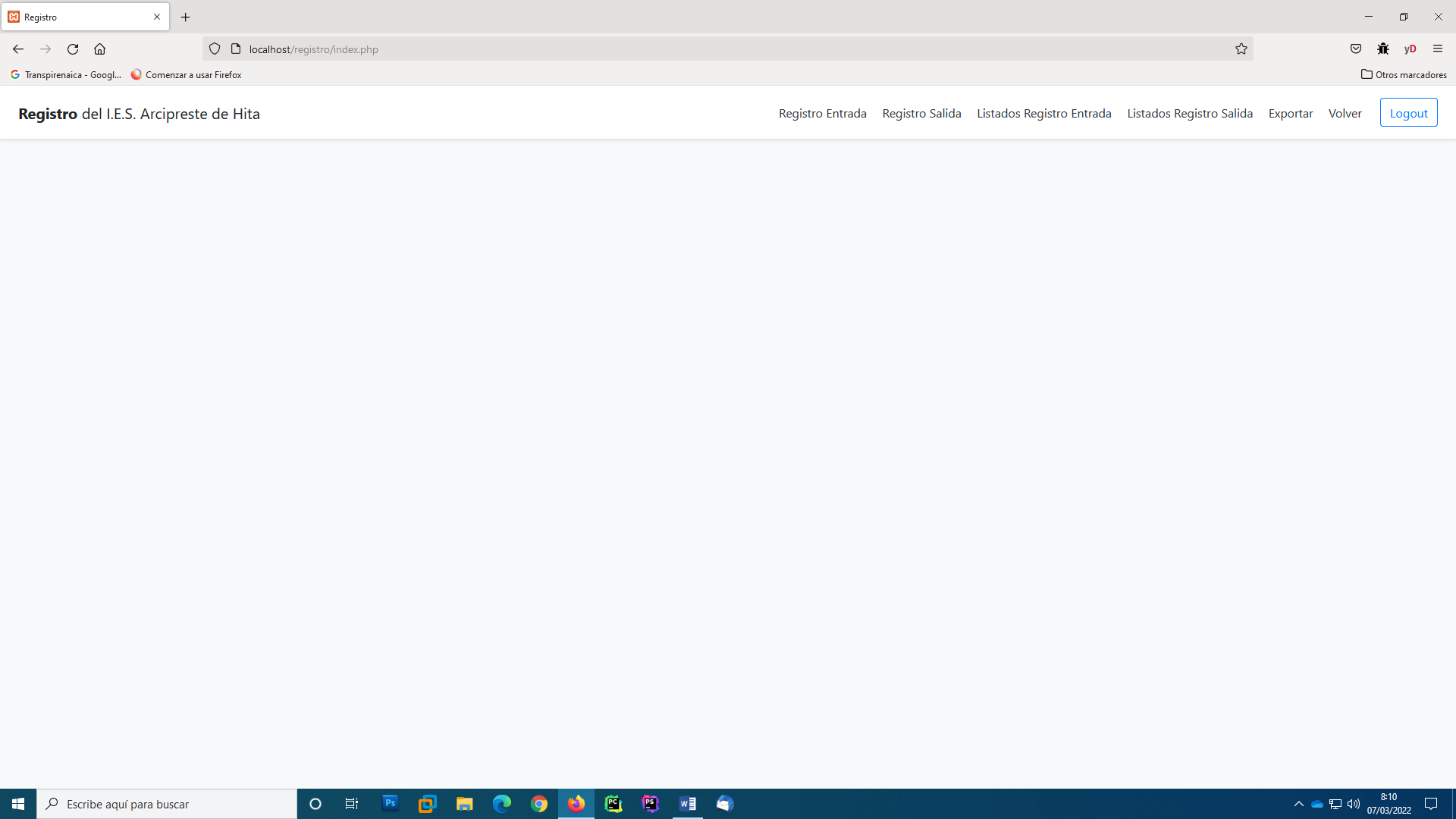


Ilustración Menú Principal

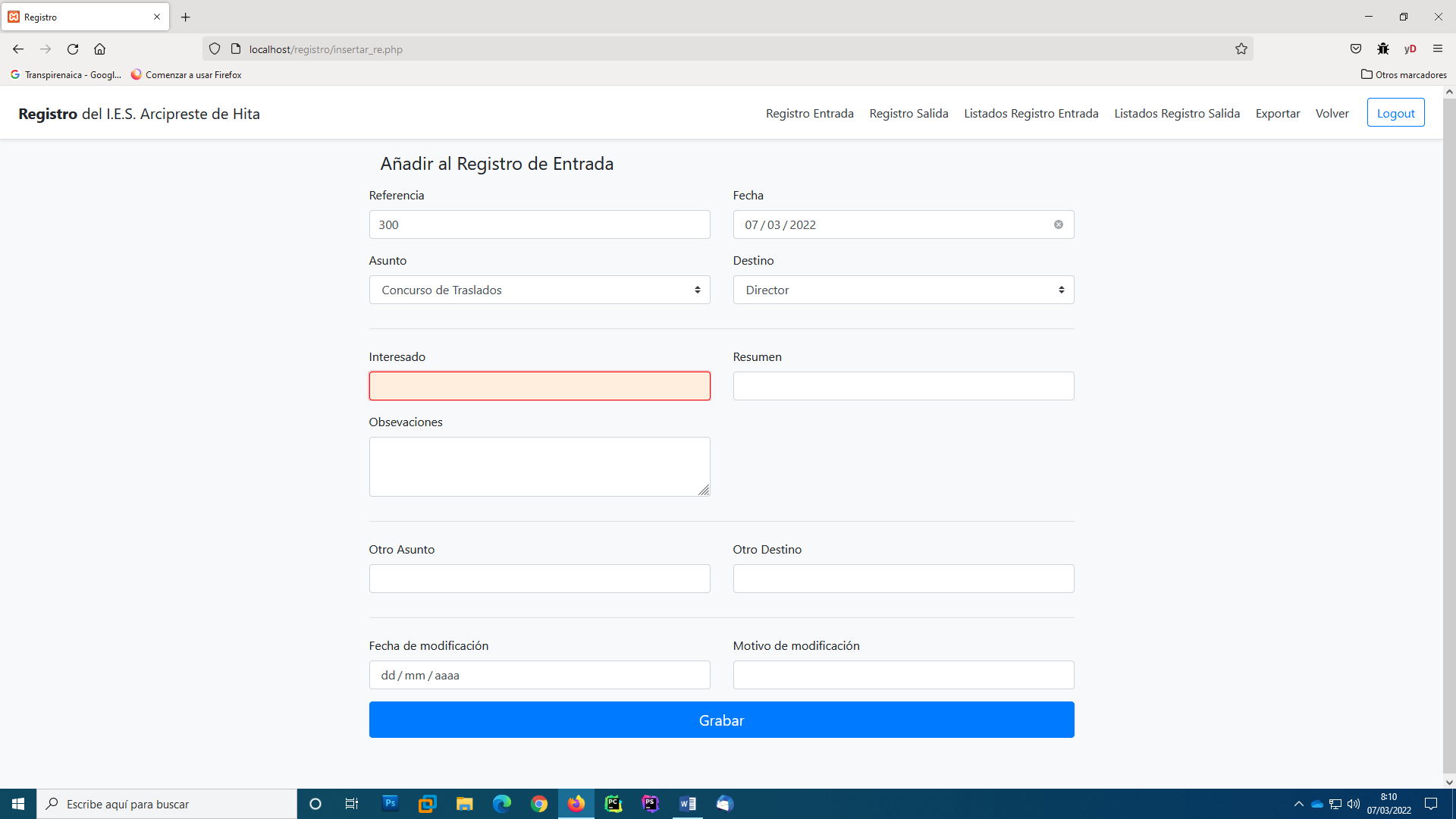


Ilustración Añadir al registro de entrada

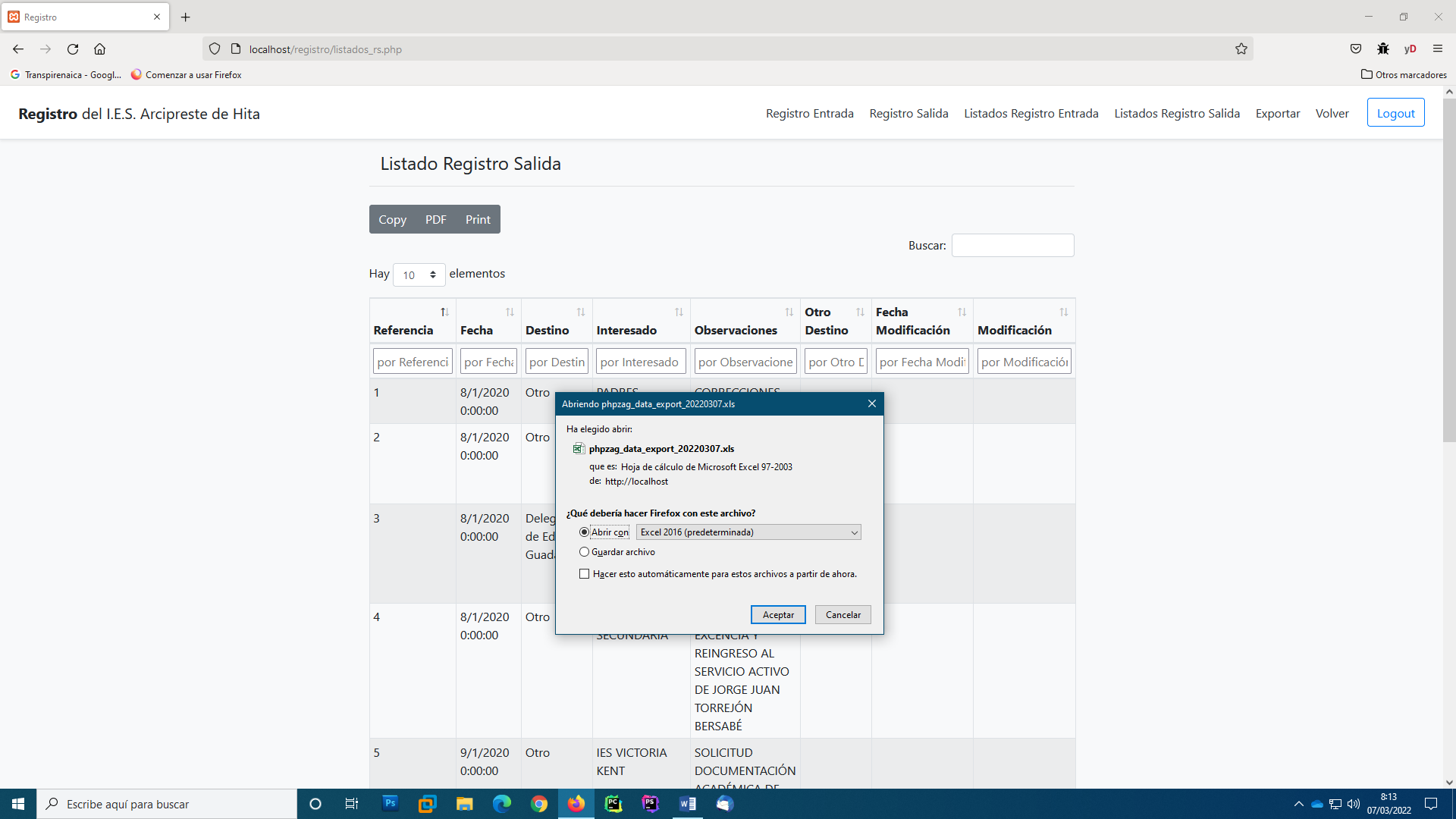


Ilustración Exportar el registro



Ilustración Añadir al registro de salida

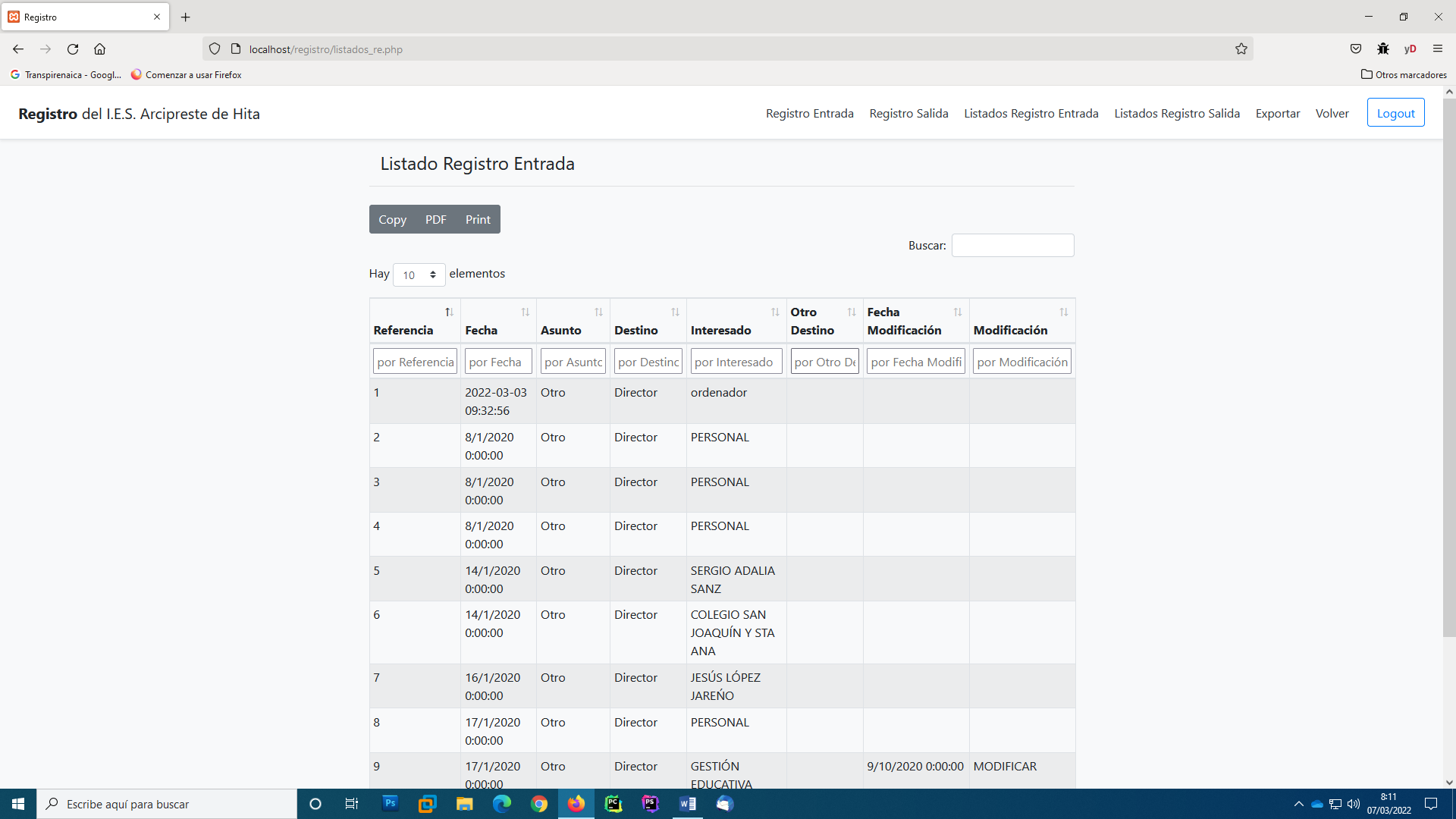


Ilustración Listado de registro de entrada



Ilustración Listado de registro de salida

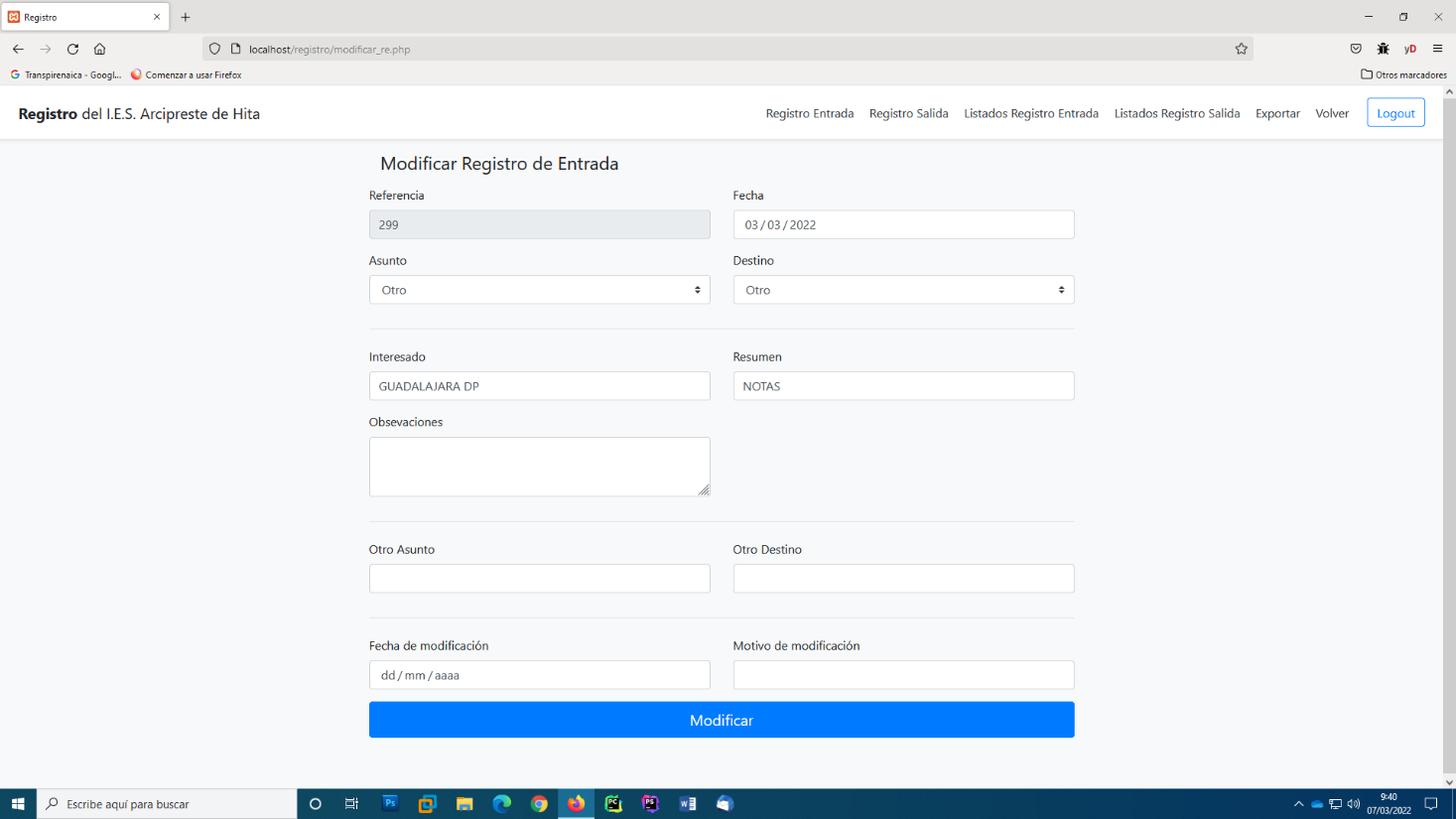


Ilustración Modificar una entrada registro entrada

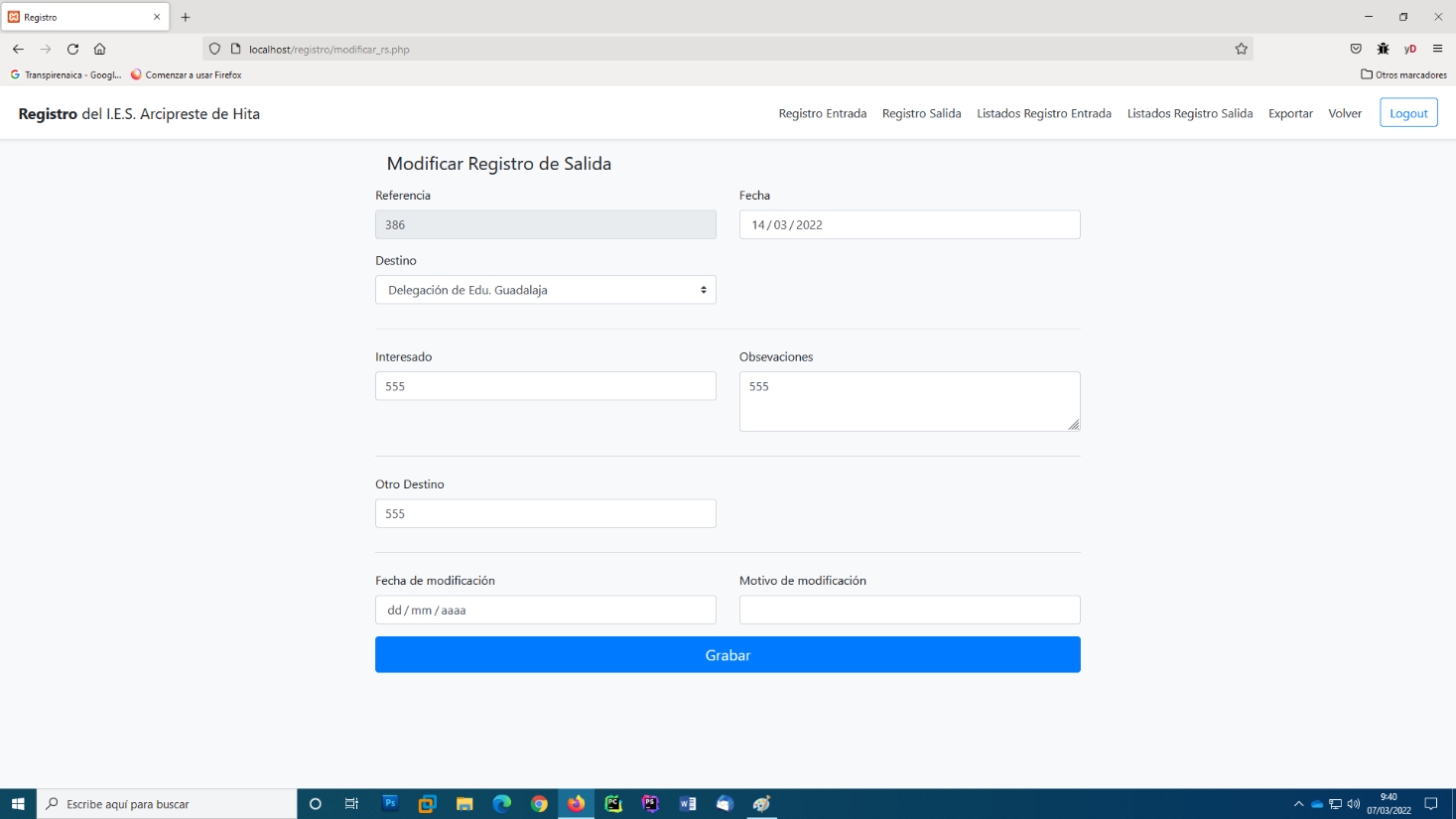


Ilustración Modificar una entrada registro salida

Desde nuestro punto de vista, un microservicio será una aplicación web mínima, autocontenida que realiza una única función. A continuación, el ejemplo que desarrollaremos en este curso.

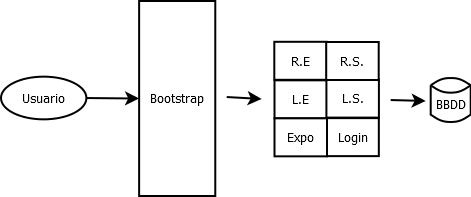


Ilustración Monolítico

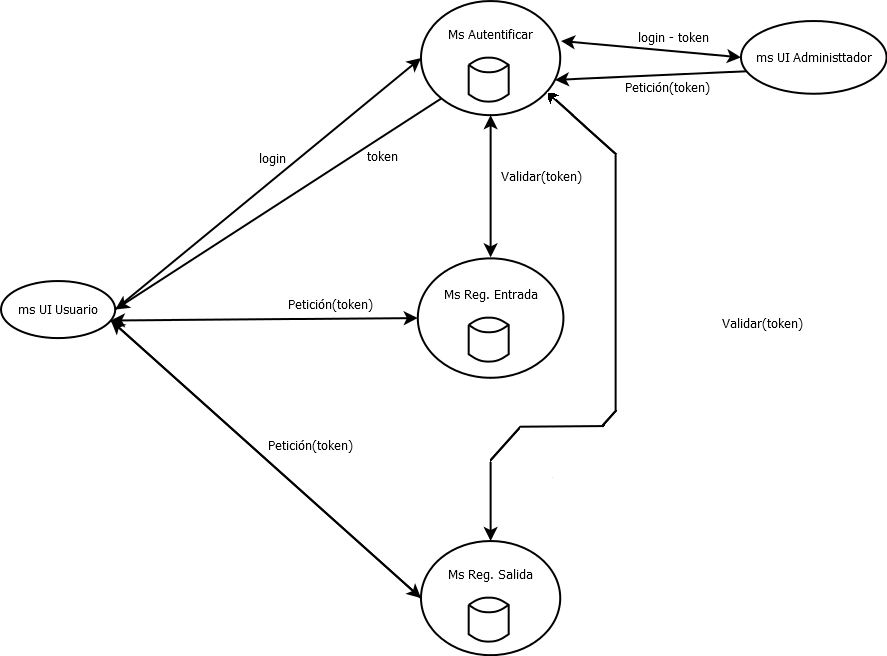


Ilustración Microservicios

Podemos observar cómo vamos a pasar de una única aplicación web con formularios y menús a un conjunto de cinco aplicaciones diferentes que van a interactuar entre ellas para dar resolución al problema planteado.

Para crear un ms de forma efectiva, deberemos tratarlo como una caja negra que espera peticiones y devuelve resultados, no siendo importante el contenido de la misma, lo único importante es el interfaz de comunicaciones que exponga el servicio y que este no cambie a lo largo del tiempo. Desde un punto de vista de la POO, un ms es una clase que encapsula una funcionalidad y lo pone a disposición de los demás a través de un API (el interfaz).

## Conceptos claves de los ms

* **Implementación independiente**. Cada ms será independiente en el sentido que podrá ser cambiado, implementado y distribuido a los usuarios sin tener que tocar nada en el resto de ms. Se busca el máximo desacoplo entre los servicios. Para poder llevar a cabo el máximo desacoplo se tienen que definir interfaces de comunicación entre los ms bien definidos y fijos, por lo que hay que hacer un análisis y diseño muy exhaustivo.
* **Mantenimiento del propio estado**. Un ms contiene todos los datos que necesita para funcionar, lo que conlleva que no hay bases de datos compartidas, cada ms tendrá la suya.
* **Tamaño**. El tamaño no es importante, debemos fijarnos en que se modele dentro del ms una única función de negocio.
* **Flexibilidad**. Queremos una estructura de ms flexible que sea capaz de resolver problemas actuales, pero también los futuros con poco esfuerzo.
* **Equipos de desarrollo amoldados a la estructura del negocio**. El patrón MVC impone que los equipos de desarrollo se crean en función de las capacidades técnicas de los componentes. Así aparece el equipo de front-end, de back-end, dba, etc. Estos equipos no tienen una visión completa del problema de la empresa, tienen una visión de la tecnología que se está usando para resolver el problema. En una arquitectura de ms, los equipos son multidisciplinares en cuanto a la tecnología y adoptan un enfoque vertical en cuanto a conocimiento del problema.

## Comparación de monolítico vs microservicios

Hemos visto que hasta hoy en día se ha desarrollado la mayoría de aplicaciones a través de un patrón monolítico (MVC), pero que ha emergido otro tipo de desarrollo como respuesta a los retos que plantea la sociedad actual.

### Ventajas de los desarrollos monolíticos

* Topología más simple.
* Flujos de trabajo más simples.
* Monitorización y testeo más simples.
* Reutilización de código más simple.

### Desventajas de los desarrollos monolíticos

* Desarrollo paralelo más complicado, al aparecer esperas entre los equipos de desarrollo.
* Poca escalabilidad.
* No se ajusta al modelo de negocio.
* Poco ajuste a cambios de negocio.
* Uso ineficiente de los recursos.
* Dependencia de la tecnología.
* Un pequeño error puede hacer caer todo el servicio.

### Ventajas de los desarrollos con microservicios

* Se pueden usar tecnologías heterogéneas, diferentes en cada ms si es necesario. Permite adaptarse a las nuevas tecnologías que vayan apareciendo o usar específicas para el ms.
* Son muy robustos, si un ms cae no debe caer el resto de ms, pero para que esta característica exista hay que asumir las características y problemáticas de los sistemas distribuidos.
* Se pueden escalar relativamente fácil. Se puede escalar solo aquellos servicios que lo necesiten, no todos.
* Al ser más simples, el desarrollo individual es más fácil pero el diseño conjunto presenta problemas que no aparecen en los sistemas monolíticos, al tener que tener en cuenta todos los problemas de las transacciones distribuidas.

### Desventajas de los desarrollos con microservicios

* El tiempo de desarrollo es más elevado que la misma aplicación monolítica.
* Es necesario que el equipo sea experto en el desarrollo de ms.
* Se tiende a creer que es necesario utilizar las últimas tecnologías, **no es necesario**, solo adoptaremos aquellas que sean necesarias. Por ejemplo, ¿para cinco servicios es necesario usar una kubernet?
* Los costes de desarrollo y mantenimiento son mayores al principio y no se van a reducir con respecto a otras arquitecturas, pero van a generar más oportunidades de negocio.
* La generación de informes del sistema será más difícil al estar toda la información repartida entre diferentes ms.
* El monitoreo y resolución de problemas es más difícil.
* La seguridad debe ser mayor, hay que asegurarse que las transacciones entre los microservicios no sean capturadas y fáciles de hackear, así como su exactitud.
* La realización de pruebas es mucho más difícil, al estar todos los ms relacionados entre sí, pero a su vez separados unos de otros. Determinar la causa del fallo será mucho más dificultosa.
* La consistencia de los datos será más difícil de mantener, al existir una BBDD distribuida entre los diferentes servicios aparecen problemas no existentes hasta ahora.

## Cómo modelar los microservicios

La definición de buenos ms se basa en determinar correctamente los límites de estos y que permitan el máximo desacoplo y aislamiento. En este sentido deberán ser servicios independientes.

* **Ocultación de la información**. Definir la información contenida en un ms y ver los límites de esos datos permitirá delimitar los “bordes” de cada ms. Deberemos minimizar el número de funcionalidades que creemos que deben tener el resto de servicios, cuestionándonos todo desde el principio.
* **Cohesión**. El código que cambia junto permanece junto, es una máxima de la programación. Por lo tanto, todo código relacionado deberá estar en el mismo ms. Si hay que cambiar una funcionalidad estará en un único sitio. Si somos capaces de definir bien los límites de manera que todo el código relacionado esté junto permitirá una mejor gestión del código.
* **Acoplamiento**. Si un ms está poco acoplado, el cambio en dicho ms no requerirá cambio en algún otro. El mejor acercamiento es que un ms conozca lo menos posible de aquellos ms con los que colabore, limitando el número de llamadas que se hacen a otros ms.
* **Interacción entre cohesión y acoplamiento**. Ambos conceptos están relacionados definiendo la estabilidad de un sistema. El sistema será estable si tienen una fuerte cohesión y un mínimo acoplamiento. Esta disposición reducirá la comunicación entre los diferentes grupos de trabajo. Para que se dé esta característica, los límites de los ms deben permanecer estables una vez definidos.

### Tipos de acoplamiento

El acoplamiento entre el código es uno de los mayores problemas que se presentan en el desarrollo de ms. Este problema no aparece en los desarrollos monolíticos de forma tan clara pero también se debe tener en cuenta.

Hay que recordar que algún tipo de acoplamiento debe existir, es imposible evitar, lo que buscamos es reducir al máximo dicho acoplamiento.

* **Acoplamiento de dominio.** Un ms hace uso de otro ms porque necesita de sus funcionalidades. Por ejemplo, un ms *Proceso de Orden*, necesitará del ms *Almacén* y del ms *Pago* para poder realizar su tarea.

Este tipo de acoplamiento es inevitable, pero hay que intentar mantenerlo al mínimo posible. Si un ms hace demasiadas funciones, o se comunica con varios ms deberemos estudiar la posibilidad de dividirlo en varios.

Este tipo de acoplamiento puede presentar problemas si la cantidad de información a compartir entre los diferentes ms es elevada. Hay que tener en cuenta que cada ms necesitará toda la información para su trabajo.

Compartir lo imprescindible, mandar lo mínimo posible

* **Acoplamiento temporal**. Se da en aquellos ms en los que la ejecución debe ser simultánea o se imponen restricciones de orden en la que uno se debe de ejecutar siempre antes que otro ms. Un ms de *Orden de Pedido* debe ejecutarse antes de la tarea del ms *Almacén*, siendo obligatorio que este segundo esté disponible cuando termine el primero para que se complete la funcionalidad.

Hay que tratar de forma cuidadosa este acoplamiento, sobre todo cuando el número implicado de ms crece y las cadenas de ejecución se agrandan.

* **Acoplamiento de transición (pass-trough).** Este acoplamiento se produce cuando un servicio pasa los datos a otro servicio simplemente porque se necesita para completar la operación más adelante en otro ms.

El mayor problema con el que nos podemos encontrar, es que si el formato de datos cambia en cualquier punto de la transmisión hay que cambiar todos los ms involucrados.

Deberemos pensar si es necesario el ms intermediario o puede ser suprimido de alguna manera para evitar este acoplamiento. Otro aspecto a tratar, son los datos a pasar al intermediario, se deben delimitar muy bien y ser lo mínimo posible.

* **Acoplamiento común.** Se produce cuando dos o más ms hacen uso de un mismo conjunto de datos o recursos (bbdd, memoria o sistema de archivos). Este acoplamiento presenta problemas con el cambio de la estructura subyacente del recurso en cuestión que implicará un cambio en todos los ms que lo usen. Este problema aumentará con la frecuencia del cambio y con el acceso concurrente al recurso por varios ms.

Para paliar el problema podemos intentar separar los recursos y que sean únicos, o en caso que no sea posible, determinar un único responsable del recurso. También se deben utilizar técnicas de bloqueo del recurso para que quede en un estado estable en caso que aparezca algún tipo de problema.

* **Acoplamiento de contenido.** Hablaremos de este tipo de acoplamiento cuando un servicio cambia el estado interno de otro ms. Por ejemplo, un ms cambia la BBDD de otro ms. Se diferencia con el anterior en que este caso la propiedad de recurso compartido es menos clara.

### Metodologías a usar

En el entorno de los ms aparecen varias metodologías que podemos usar para crear una estructura. La primera aproximación es el **diseño basado en dominios**.

La idea es determinar funcionalidades o “agregados”, entendidos como estructuras que tienen estado. Los agregados serán tratados por los ms sabiendo que un agregado deberá estar en un solo ms servicio, pero un ms podrá atender a varios agregados. Los agregados definen cada una de las funciones básicas que implementarán la función de negocio.

Los agregados son datos autocontenidos que representan un estado de la máquina que se centra en un solo concepto del dominio. Con estos agregados hay que determinar los límites que representan el conjunto de agregados relacionados y el interfaz a exponer fuera del ms. Estos límites se buscarán refinando uno inicial, minimizando lo más posible el mismo.

El proceso se podría resumir de la siguiente forma: se identifica los eventos del dominio (un problema de negocio a solucionar, por ejemplo: el almacén), estos eventos representan sucesos en el sistema. Después se identifican los comandos que causan dichos eventos. Los comandos determinan las decisiones que toman las personas para hacer algo. Los eventos son la semilla de los agregados, reorganizaremos los comandos y eventos, relacionándolos entre sí para generar el agregado. En una última fase se agruparán los agregados en torno a un ms.

La metodología impone un flujo de trabajo, pero no asegura un buen resultado. Deberemos tener en cuenta algunos factores más para un buen diseño.

* **Volatibilidad.** Determinar las partes que más cambian de un sistema permitirá extraer funcionalidades (eventos y comandos) para crear los agregados.
* **Datos.** Considerar la naturaleza de los datos nos permite detectar diferentes formas de descomposición para los ms e intuir los límites.
* **Tecnología.** El uso de diferentes tecnologías nos puede a ayudar a determinar los límites entre los ms.

# Capítulo II. Conceptos de desarrollo

## Partiendo de un desarrollo anterior

Generalmente partiremos de una aplicación existente que deseamos migrar a un entorno basado en ms. Este sistema seguirá, lo más probable, una estructura monolítica que podremos utilizar para crear el nuevo sistema. En este punto vamos a tratar los aspectos más importantes en este tipo de migraciones ya que son las más frecuentes.

### Tener un objetivo claro

El crear un sistema con ms no es el objetivo, los ms son la herramienta para conseguir lo que queremos. Solo empezaremos a pensar en migrar si no encontramos otra solución más fácil, o una arquitectura basada en el patrón MVC que nos sirva para conseguir el objetivo. Parece una incongruencia el recomendar no usar ms a menos que sea necesario en unos apuntes de creación de ms, pero las arquitecturas basadas en ms son muy complejas con necesidad de programadores muy experimentados en el desarrollo de ms.

En todo caso, es imprescindible tener un objetivo definido o será muy difícil saber por dónde empezar a diseñar la estructura de ms. Hay que tener en cuenta que esta estructura se basará en soportar una estructura de negocio, no una aplicación para resolver un problema, con lo que su ámbito es mucho mayor que el necesario en un desarrollo monolítico.

### Migración incremental

Se dice en la industria del entretenimiento que el show siempre debe continuar. Pues esa máxima la deberemos aplicar a nuestras migraciones. No podemos dejar a la empresa sin una aplicación que está usando actualmente para hacer nuestra migración, por lo que deberemos pasar de un sistema monolítico a otro basado en servicios mediante sucesivas modificaciones del sistema, en la que irán desapareciendo paulatinamente cada vez más partes del modelo monolítico que serán sustituidas por partes del modelo de ms, hasta el estado final en el que solo quede el sistema basado en ms.

Una posible aproximación es crear en cada paso un ms que sustituya solo una parte de la aplicación monolítica. Esta aproximación permitirá aprender cada vez más del propio sistema para los pasos posteriores y limitará los daños en caso que algo no funcione de forma adecuada, siendo relativamente sencillo revertir el sistema a su estado anterior.

Durante la migración nos tendremos que plantear si es imprescindible eliminar todo el código monolítico, si es necesario para nuestros objetivos. El objetivo nunca debe ser eliminar el código MVC, debe ser soportar nuestra función de negocio.

### Estudio detallado del dominio del problema

Hay un peligro muy grande al diseño ms, el intentar dicho diseño sin tener un conocimiento claro del dominio del problema. Si existe una aplicación previa en la empresa tendremos una muy buena fuente para estudiar dicho dominio. Es imprescindible conocer el problema al que nos enfrentemos de forma exhaustiva para poder realizar un análisis y diseño adecuado.

En este estudio del dominio y apoyándonos en las aplicaciones existentes, descubriremos aquellas partes de la aplicación que cambian más rápidamente y ver si se pueden implementar como ms. También atenderemos a la viabilidad de los ms antes de implementarlos, no podemos lanzarnos a desarrollar un ms sin antes estar completamente seguros de que se puede realizar. La decisión final de si una parte de la aplicación se convierte o no en ms será un balance entre lo fácil de extraer de la aplicación monolítica contra los beneficios.

Es aconsejable empezar por algo pequeño y fácil, adquirir habilidades en la programación de ms y del dominio del problema y continuar con retos más difíciles.

### Descomposición por capas

Una vez detectado y extraído el primer servicio, nos preguntamos cómo continuamos. Podemos usar las capas del patrón MVC si existe en nuestra aplicación monolítica y tomarla de base.

* La capa de vista no tiene una correspondencia uno a uno con los ms. No se puede ignorar el GUI al migrar ya que a menudo el cambio del interfaz a un sistema basado en ms aporta grandes beneficios, pero no veremos dichos beneficios hasta que el resto de ms estén creados.
* El back-end (modelo). Tenemos que recordar que se debe mantener junto todo el código relacionado, así como los datos correspondientes, en caso que no sea posible es mejor no realizar la migración de dicho servicio.

La descomposición por capas puede ayudar a comenzar la descomposición en los diferentes ms, pero siempre intentaremos sacar los datos primero para identificar aquellos casos en los que no se puedan sacar los datos de forma clara.

### Patrones de diseño para la descomposición

Los casos que se dan a la hora de convertir un sistema monolítico en uno orientado a ms suelen estar muy definidos. Estos casos de uso comunes se implementan a través de patrones de diseño. Dentro de los más comunes tenemos.

* **Crear un punto intermedio** que intercepte todas las llamadas al servicio, tanto el nuevo como el viejo, distribuyendo estas llamadas entre ambos. El interceptador determinará cuantas se mandan a cada servicio para por determinar posibles fallos y reconducir el servicio sin problemas.

Intercepción

Viejo

Nuevo

* **Ejecución paralela**. Se ejecutan ambos ms a la vez para evitar problemas hasta que estamos seguros que el nuevo puede sustituir completamente al viejo.
* **Cambio de función**, con la que podemos habilitar o inhabilitar un ms de forma independiente, o intercambiar entre dos posibles implementaciones del mismo ms.

### Consideraciones sobre los datos

* **Rendimiento**. Las BBDD relacionales son buenas uniendo tablas, pero al dividir los datos en ms estamos separando los datos en varias BBDD, con lo que hacer estas agregaciones puede ser complejo y lento, llevándonos a posibles problemas de latencia.
* **Integridad de los datos**. Las BBDD son buenas asegurando la integridad de los datos, al estar separados en varios ms, no podemos asegurar dicha integridad. Se podría paliar el problema en caso de borrado usando marcadores en vez de realizar las transacciones de forma inmediata.
* **Transacciones**. El modelo basado en ms es un modelo distribuido, con lo que las transacciones de datos de las BBDD relacionales no se pueden ejecutar, hay que tener en cuenta todo lo relacionado a transacciones distribuidas que son difíciles de implementar y no tienen la misma seguridad que las locales.
* **BBDD de Recopilación**. Idealmente cada ms debe tener su propia base de datos, pero se darán casos en lo que no sea posible. En este caso, deberemos crear una BBDD externa al ms, y posiblemente a varios en donde se gestionarán los datos. Este modelo implica romper con la filosofía general y se recomienda que solo se almacenen en esta BBDD los datos imprescindibles, evitando trasladar todos los datos a la BBDD externa.

## Comunicaciones entre los ms

La estructura que estamos creando es básicamente una estructura distribuida basada en una red de comunicaciones. Como tal, es imprescindible que atendamos en gran medida a cómo establecer las comunicaciones entre los diferentes ms, así como a asegurar dicha comunicación y la tolerancia a fallos tanto de la red como de cada uno de los componentes. Los aspectos más relevantes que tenemos que tener en cuenta para la migración son:

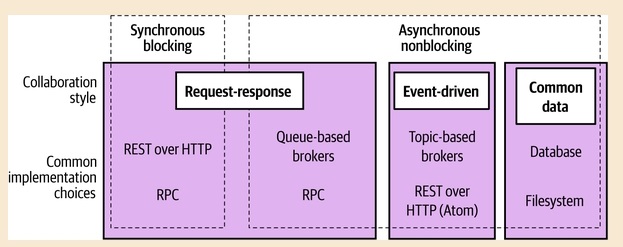
* **Rendimiento.** El rendimiento se degrada al tener que usar una red de comunicaciones para el paso de información entre los ms. Además, aparecerán nuevas cuestiones que antes no existían, como son la latencia de la red, la tolerancia a los fallos y la sobrecarga de datos que generan estas transferencias.

Otro elemento a tener en cuenta es la serialización de los datos. En un sistema monolítico los datos se pasan a través de la memoria o ficheros, pero es el propio intérprete el encargado de la gestión y conversión de los mismos. Ahora somos nosotros los que tenemos que definir un estándar de comunicación y convertirlos a dicho estándar antes y después de viajar por la red. Qué estándar elegimos: XML, JSON, YAML, otros.

También tenemos que tener en cuenta la cantidad de datos que hay que transmitir entre los ms. Si es muy elevada el tiempo de transferencia y de procesamiento serán también puntos a tener en cuenta en nuestro desarrollo.

* **Cambio en los interfaces.** Uno de los puntos más críticos de un ms es su interfaz. Este es elemento de comunicación entre el ms y el exterior. Si se producen cambios frecuentes en el interfaz o los datos que necesita, implicará que todos aquellos servicios que lo utilicen deben cambiar también. Por tanto, hay que minimizar el cambio de interfaces y en caso de ser necesario, utilizar algún patrón de diseño para la descomposición buscando minimizar problemas.
* **Manejo de errores.** Los errores van a ocurrir. Antes aparecían asociados al usuario y a la programación del servicio (deterministas y fáciles de detectar), ahora además hay que añadir los problemas que puede causar la infraestructura (hardware y software) implementada subyacente. Los problemas más comunes que nos vamos a encontrar son: que no llegue una petición y hay que volver a lanzarla; Nunca llegue una respuesta; etc. Si creamos una forma de tratar los errores adecua haremos que la gestión de los mismos sea más fácil. Por ejemplo, implementar algo similar a los errores http.

### Tecnologías de comunicación



Para poder implementar una tecnología de comunicación debemos investigar todas las opciones posibles, evaluarlas y decidir la que más se ajuste a nuestras necesidades. Pero tendremos en cuenta que generalmente se usa una mezcla de diversas tecnologías, entre las que pueden aparecer:

* **Bloqueo síncrono**. Un ms hace una petición a otro ms y se bloquea a la espera de la respuesta. Se utiliza cuando debemos estar seguros de que la llamada se hace y tiene una respuesta, sino hay que rehacerla. El problema de este método es el acoplamiento que genera entre ambos servicios.
* **Llamada asíncrona**. La llamada del ms origen es no bloqueante. Los ms se desacoplan temporalmente, con lo que la respuesta puede llegar muy separada en el tiempo. La mayor dificultad que nos encontramos en este tipo es la complejidad.
  + **Datos comunes**. Qué pasa si un ms pone datos en un área compartida con otro ms y este segundo lo recoge de esa área. Para usar este mecanismo es necesario algún tipo de servicios persistente: BBDD, Sistema de ficheros. En tal caso hay que tener en cuenta la estructura de los datos que se almacenarán y generará acoplamiento entre ambos. Además, añade la cuestión de la señalización al ms consumidor, cómo se le dice que hay datos esperando. Por último, hay que tratar también el hecho que el sistema intermedio de almacenamiento no esté disponible, en tal caso cómo continuará nuestro sistema.
  + **Petición – respuesta.** Para una implementación asíncrona podemos usar colas compartidas. Si es el caso, los ms tienen que ser notificados por la cola para una comunicación efectiva entre ambos y tiene que haber alguna manera de enrutar los mensajes hacia los ms desde las colas. También se debe asegurar que las respuestas puedan llegar mucho más tarde que cuando sean creadas, pueden tardar incluso horas, pudiendo no estar disponible el ms que lanzó la petición y haber sido remplazado por otra copia. Además, hay que tratar el hecho de que un ms necesite realizar varias peticiones simultáneas, en tal caso hay que decidir si se hacen una a una o se lanzan en paralelo con la consiguiente complejidad.
  + **Dirigido a eventos**. En este modelo un ms emite un evento que será tratado por otro ms. Se debe tratar cómo emitir el evento, cómo los consumidores serán notificados o encontrar los eventos. Este tipo es muy difícil de crear y mantener, pero una vez establecido es muy eficiente.

## Implementación de la comunicación

Hemos hecho un pequeño resumen de los posibles modelos y de las implicaciones de los mismos, en este apartado enumeraremos las posibles tecnologías que podemos usar para esta implementación.

En todo momento debemos tener en cuenta una serie de principios básicos para implementar una comunicación efectiva independientemente de la tecnología o tecnologías que usemos. Primero haremos que la compatibilidad hacia atrás sea fácil, haciendo el interfaz explícito, claro y sin cambios. Segundo, mantener el API independiente de la tecnología subyacente, nos permitirá mayor flexibilidad de cambios. Tercero, hacer el ms lo más sencillo para los potenciales clientes, no obligando a usar una tecnología en concreto para conectarnos al ms. Y, por último, ocultar la implementación interna, solo mostrando el API y los resultados.

### Opciones tecnológicas

* **Llamadas a procesos (RPC; SOAP, gRPC).** Esta opción se debe usar cuando queramos tener un gran control sobre el cliente.
* **API REST.** Se utilizará para comunicaciones síncronas generalmente y acceso desde múltiples clientes, tanto ms como externos. En este caso hay que investigar las nuevas tecnologías emergentes en protocolos, tanto HTTP/3 como QUICQ sustituto de TCP.
* **GrapQL.** GraphQL es un lenguaje de consulta y tiempo de ejecución del servidor para las interfaces de programación de aplicaciones (API); su función es brindar a los clientes exactamente los datos que solicitan y nada más. Gracias a GraphQL, las API son rápidas, flexibles y sencillas.
* **Message Brokers.** Es una aplicación middleware. Se suele usar para la comunicación asíncrona entre los ms. Aseguran la entrega de los mensajes.

### Formatos de la serialización

Ya comentado en un punto anterior, es la necesidad de definir el esquema de serialización de nuestros datos a la hora de viajar por la red. Entre los diferentes formatos que tenemos están:

* **Basados en texto.** JSON, XML o AUro. Cada uno tiene sus ventajas e inconvenientes, deberemos estudiarlos detenidamente y determinar el que vamos a usar.
* **Binarios.** En este caso se mandan los datos en forma binaria, solo lo usaremos en ms que tengan y necesitemos muy baja latencia.

### Esquemas del API

A la hora de generar un esquema para el API valoraremos la inclusión en el mismo de la versión del mismo, proporcionándonos de esta manera una forma fácil de detectarla. Así podremos encontrar esquemas del tipo: **mi\_servicio/v1/admin/función**.

### Manejo de los cambios en los ms

Unos de los procesos más complejos es la migración de un ms a otro ms que haga la misma función, por algún tipo de cambio, ya sea interno o de API. En estos casos debemos tener un gran cuidado para evitar problemas en el sistema completo.

* **Sin cambio en el API.**
  + Añadiremos elementos al ms sin eliminar ninguno, el nuevo ms puede asumir todas las funciones del viejo sin cambio en los clientes.
  + Seremos flexibles en los tipos de datos que el ms consume.
  + Realizaremos APIs explícitas.
  + Realizaremos los cambios en los ms lo antes posible, en cuanto sea detectada la necesidad, no esperaremos a tenerlo completo.
* **Con cambio en el API.**
  + Cambiar a la vez el ms cliente y el ms servidor.
  + Permitir la coexistencia de varias versiones del mismo ms servidor.
  + Emular el API antiguo en el nuevo a través de PROXIS.

### Principio DRY (Don’t Repeat Yourself)

Desde el comienzo de la programación aparece una y otra vez este principio, que en general dice que no debemos repetir código o funcionalidades en diferentes archivos o servicios. Atendiendo a esta máxima atenderemos a:

* Evitar código repetido.
* Evitar funcionalidades repetidas.
* En un ms, nos llevará a acoplar el ms, mejor que repetir código en cada ms. Se puede plantear crear una librería con código que copiaremos en todos los ms que la necesiten.
* En caso de usar la misma librería en diferentes ms, no actualizar todos los ms a la vez, evitando que si existe un fallo se extienda a todos.

### Descubrimiento de servicios

Para que dos servicios se puedan comunicar deben conocerse ambos y el mecanismo que tienen un ms para saber de otro ms se denomina descubrimiento. Los esquemas que podemos usar varían desde los más simples (configuración al arrancar el ms) hasta basados en servicios de registro, la decisión variará de las necesidades del negocio. En cualquier caso, si el número de ms crece, o el número de instancias de cada ms es variable, tendremos que implementar un servicio de descubrimiento. Este servicio se gestiona en dos partes, una primera se registra el ms que ofrece sus servicios y en la segunda parte un ms cliente busca un ms que pueda atender su petición.

Las tecnologías son diversas y van aumentando con el tiempo. Entre las más comunes tenemos el propio sistema DNS (pero es poco flexible) y servicios de registro dinámico (Zookeeper, Consul, Kubernetes).

### Gestión del crecimiento

Si nos hemos embarcado en un sistema de ms es muy probable que esperemos el crecimiento exponencial de nuestro sistema, con la consiguiente pérdida de rendimiento al no ser posible tratar el número de peticiones con el número actual de ms. Para poder evaluar y gestionar el crecimiento tendremos en cuenta en dónde se está dando dichas necesidades, dentro de la red o fuera de ella.

Así definiremos el tráfico norte-sur, como el tráfico que se produce entre los límites de nuestro sistema y el mundo exterior. Definiremos el tráfico este-oeste el que se produce internamente en nuestro servicio por la comunicación entre los ms. Es determinante saber cuál es el tráfico que ha crecido para poder tomar las decisiones adecuadas de crecimiento. La ventaja en un sistema basado en ms es que podemos ver si es un ms en concreto el que necesita crecer, diseñando soluciones tecnológicas fáciles para acomodar dicha carga.

Si la carga se produce en el tráfico Norte-Sur deberemos adoptar algún tipo de API Getway para gestionarlo. Si la carga se produce dentro del sistema tendremos que aumentar la malla de servicios para soportar la carga.

### Servicio de documentación

Según van creciendo el número de ms y las relaciones entre ellos, es más complejo mantener una documentación adecuada de los mismos. Además, los equipos tienden a infravalorar la creación de una documentación adecuada, postergando este paso siempre al final del proceso, que muchas veces se solapara con nuevos cambios a introducir en nuestro ms.

Es imprescindible mantener la documentación al día. Se pueden usar mecanismos de documentación automática que generarán toda esta documentación y no son dependientes de que sea creada por los programadores.

# Capítulo III. Implementación

## Transacciones

Uno de los conceptos más difíciles de implementar son las transacciones entre los diferentes ms. Tanto en las peticiones como en la gestión de los datos.

En las BBDD relacionales, las transacciones cumplen con un conjunto de características que se denominan ACID. Es decir: Atómicas, Consistentes, Aisladas y Durables. En este caso una transacción en una BBDD relacional asegurará la integridad de los datos.

Los ms dividen los datos entre cada uno de ellos, creando una verdadera BBDD distribuida, por lo que no existe la posibilidad de crear transacciones ACID en este tipo de sistemas.

Para intentar solventar las transacciones distribuidas, se implementa un protocolo de dos fases. En la primera fase todos los implicados votan si el cambio a realizar es posible y en la segunda fase se lleva a cabo dicho cambio. Este sistema no asegura que una transacción cumpla con la norma ACID, ya que la latencia de cada ms puede ser diferente y hacer que la integridad de los datos no sea correcta durante el tiempo de cambio. Este protocolo lo usaremos solo si es imprescindible, e intentando que sea un único ms que lo utilice.

Visto que el protocolo anterior no funciona, aparece SAGAS. Este sistema divide la transacción en partes y hace que cada parte se haga de forma independiente. El problema es que no proporciona atomicidad en la transacción, pero cada sub-transacción nos da la suficiente información para saber cómo va la transacción general y si hay algún fallo gestionarlo.

* **Fallos.** Los fallos en una sub-transacción se pueden gestionar de dos maneras, backwards o foreward. En el primer caso se deshace todo lo hecho hasta el momento (incluso lanzando órdenes que lo deshagan), en el segundo caso se reintenta el fallo para seguir la transacción.
* **SAGAS** solo puede arreglar fallos de la lógica del servicio, si falla el hardware habrá que tratarlo de otra manera.
* **RollBack.** En caso de error, hay que crear transacciones compensatorias para la sub-transacciones que se ha realizado ya.

## Gestión del código

A la hora de programar debemos tener en cuenta cómo se va a realizar el desarrollo y cómo se va a mantener el código para los equipos de desarrollo. Ambas cuestiones presentan diversas soluciones que deberemos estudiar y adaptar para nuestra empresa.

### Desarrollo del código

A la hora de crear el código podemos utilizar varios enfoques:

* **Integración continua.** Se mantienen todos los ms sincronizados con el nuevo código. Todo código nuevo se integra con el existente en cuanto está desarrollado, pasando por las fases de test. Debemos preparar un conjunto de test para realizar antes de la integración del código. En caso que se detecte un error de integración, se deberá solucionar lo antes posible. En este modelo no se recomienda el uso de “ramas” de desarrollo ya que retrasará la integración en el código principal, en caso de ser necesarias, se usarán con partes pequeñas de código, no con ms completos.
* **Desarrollo continuo.** En este caso dividiremos los test en varios bloques, de los más específicos a los más generales, haciendo primero los test que corran más rápido para detectar los fallos cuanto antes y arreglarlos.

### Organización del código

El código tiene que estar depositado en algún lugar, en ese sentido es imprescindible definir una política de almacenaje del mismo, para evitar duplicidades y posibles pérdidas del mismo. En este sentido podemos adoptar dos aproximaciones: de repositorio único y de múltiples repositorios.

En caso de repositorio único, se debe también “compilar” todo el código con la pérdida de tiempo correspondiente. A priori parece una solución para pequeños proyectos, pero grandes empresas como Google han adoptado este enfoque.

En caso de múltiples repositorios, se podrá crear un repositorio por servicio o uno por equipo, siendo responsable un único grupo de personas en ambas aproximaciones.

### Propiedad del código

No se refiere este apartado a propiedad “legal” o “intelectual” que está claro que es la empresa para la que desarrollamos el software, se refiere a responsabilidad de mantenimiento y control de cambios sobre el software.

En este caso, la propiedad se puede definir en términos de dura, débil o de todos. En caso de una propiedad dura, el equipo será responsable de todos y cada uno de los cambios a introducir, independientemente que dichos cambios provengan del propio equipo o de equipos externos. Si definimos la propiedad del código como débil el equipo será responsable de sus cambios, pero también de revisar y admitir los cambios propuestos por equipos externos. Por último, en caso de usar la propiedad de todos, implicará que cualquier desarrollador de cualquier equipo, independientemente de su procedencia, podrá cambiar el código del ms.

## Despliegue

El despliegue es el proceso de llevar a producción un ms. Este proceso debe ser controlado y delimitado para evitar posibles problemas en el mismo. Además, debe de ser fácilmente reversible en caso que el despliegue falle. Tendremos en cuenta los siguientes aspectos:

* Pueden existir varias copias del mismo ms y eso complicará las comunicaciones entre los ms existentes y los nuevos. En este caso hace falta un balanceador de carga.
* Hay que asegurar la disponibilidad distribuyendo la carga de los ms entre varios servidores.
* La BBDD nos presentará problemas en caso que existan diferentes copias del mismo ms. En caso que sea local al ms se gestionará mal, en caso que sea global hay que asegurar transacciones ACID.
* Una misma infraestructura de BBDD podrá servir a varios ms, deberemos tratar la BBDD como un servicio más, dotándolo de posibilidad de réplicas y balanceo de carga. Hoy en día los servidores más comunes permiten ambas configuraciones. Pero además aparece un nuevo problema relacionado con el despliegue. Van a convivir modificando la BBDD dos ms que realicen la misma función, con lo que en principio accederán a la misma BBDD. Este punto presenta problemas en caso que se de algún fallo en el nuevo ms y haya que deshacer las transacciones lanzadas.
* Los desarrollos complejos implementan entornos de desarrollo. Estos entornos se suelen nombrar como “desarrollo”, “pruebas” y “producción”. El problema con esta estructura es que los datos de los entornos y de producción no son los mismos, ni las arquitecturas hardware subyacentes tampoco, por lo que se pueden dar problemas a la hora de desplegar nuestro ms a producción. El entorno de “test” debe ser lo más parecido con el de “producción”, tanto a nivel de datos, como a nivel de arquitectura física.

Para hacer un buen despliegue de nuestra aplicación basada en ms deberemos de tener en cuenta los siguientes principios básicos:

* Se debería desplegar cada ms en su propia máquina, de tal manera que si cae no impacte significativamente en el resto de ms.
* Se elegirá una infraestructura que permita el despliegue automático de los ms.
* Desplegaremos nuevos ms teniendo en cuenta que nunca debe dejar de funcionar el sistema, con lo que haremos estos despliegues con cuidado y muy planificados, nunca sustituyendo en un primer intento el ms antiguo completamente.

Independientemente de todas las consideraciones anteriores, cada servicio debe correr en algún tipo de máquina o software emulador para llevar a cabo su tarea. Dentro de las arquitecturas que podemos elegir para ejecutar los ms encontramos:

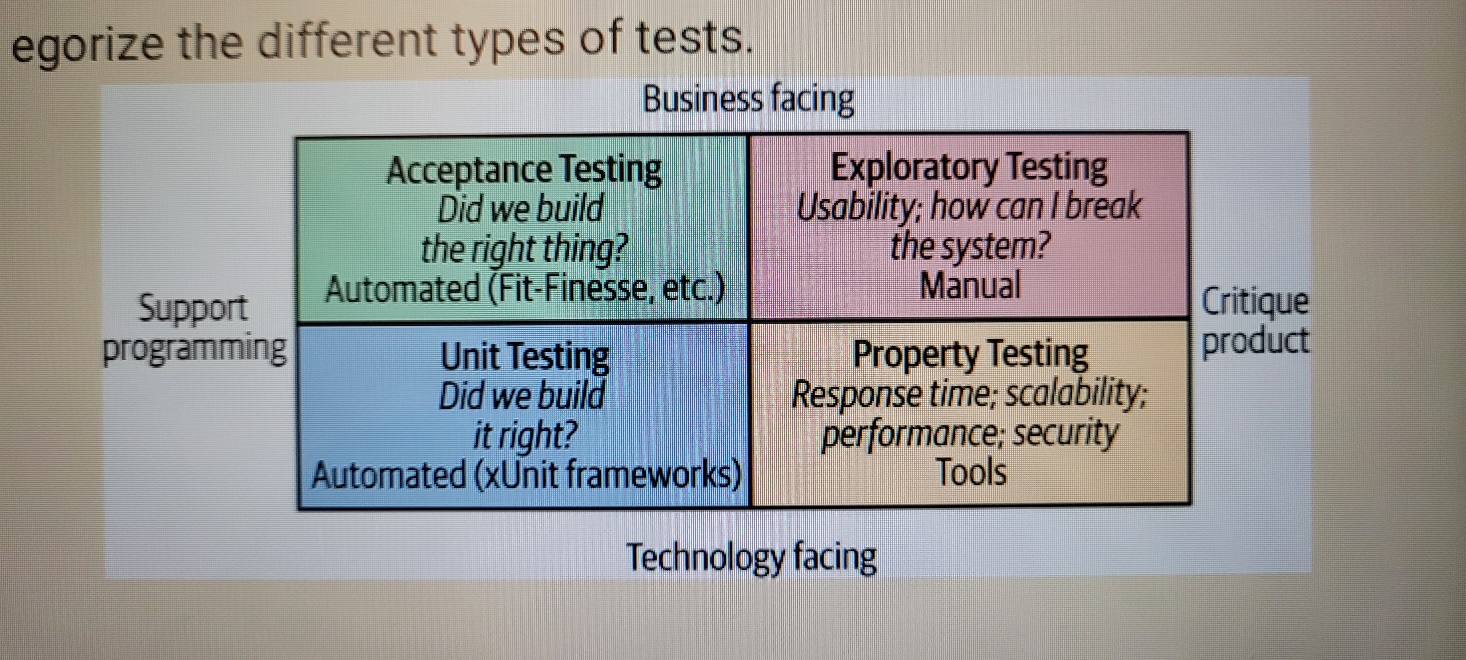
* Máquinas físicas, una por ms o una para varios ms.
* Máquinas virtuales.
* Contenedores. Docker.
* Contenedores de aplicaciones.
* Plataformas como servicio (AWS beanstalk)
* Funciones como servicio (Azure function)

La elección de la plataforma hardware para nuestro ms dependerá de muchos factores, entre los que encontramos nuestra capacidad de un departamento de desarrollo y gestión de software, la existencia de un departamento de IT, la existencia de hardware apropiado, tamaño del sistema, carga esperada a servir por el sistema y muchos más factores.

Por último, hablaremos de cómo desplegar nuevo código a nuestros usuarios. No es una tarea fácil ya que la disponibilidad del servicio no puede verse comprometida y tiene que seguir funcionando antes y después del despliegue. Para una gestión eficiente del despliegue de nuevas (o la sustitución) de funciones en nuestro sistema, nos podemos basar en las siguientes opciones:

* Poner ambas funcionalidades, la antigua y la nueva tras un interruptor y activar una u otra función a nuestro deseo. En caso de fallo revertimos al sistema antiguo rápidamente.
* Despliegue por centinelas. Podremos activar la nueva funcionalidad para varios clientes, probarla y en caso de fallo revertirla rápidamente. En caso de acierto aumentar la cartera de clientes a la que está dirigida para comprobar como escala y seguir hasta habilitarla a todos los clientes.
* Ejecución paralela. En este caso la petición es recibida por un proxy que la redirige a ambas versiones, comprobando el resultado de ambas y las compara.

## Test de la aplicación



Si en un desarrollo monolítico el testeo de nuestra aplicación se hace imprescindible, en un desarrollo bajo ms mucho más. Como ya se ha indicado este tipo de desarrollo distribuido introduce nuevos retos y dificultades a la hora de la creación, pero también lo hace en el momento que queremos comprobar si nuestro sistema funciona correctamente. Hay que tener en cuenta que ahora en vez de un único proceso, tendremos múltiples y trabajando en paralelo, en varias máquinas con lo que el número de fallos y el tipo de ellos aumentan considerablemente.

En este entorno lo primero que se tiene que hacer es caracterizar los diferentes tipos de pruebas que se pueden llevar a cabo en los sistemas basados en ms. Dentro de estas se encuentran:

* Pruebas de unidad. Permiten asegurar que una función o método se ejecutan tal y como fueron diseñados. Este tipo de pruebas suelen ser muy rápidas y la razón principal es el tener un feedback casi inmediato sobre los posibles errores.
* Test de servicio. Usando el API desarrollado de un ms, comprobaremos todo el funcionamiento del mismo.
* Test de aceptación. En este caso las pruebas se correrán sobre todo el sistema completo para comprobar su funcionamiento. Nos aseguran que el sistema cumple con los requerimientos de diseño pactados con el cliente.

Todos los sistemas deben probar y se utilizarán los tres tipos de pruebas, pero tendremos que mantener un equilibrio entre las mismas para compensar el tiempo de ejecución de las mismas con la rápida detección de fallos. Así las primeras serán más abundantes y se correrán más veces que las últimas. Es muy importante desarrollar un plan de test adecuado para nuestro sistema.

### Implementación del test de aceptación

No cabe duda que uno de los puntos críticos de nuestro desarrollo es el test de aceptación. En este momento el cliente verifica y acepta el desarrollo del sistema. Por tanto, es imprescindible que se establezca un buen plan para dicho test y su implementación sea correcta.

El objetivo de estos test es comprobar que el sistema funciona de forma correcta. En concreto comprobaremos que el API desarrollado por los ms es el indicado y se comporta de la forma esperada. En este tipo de pruebas el despliegue de los ms que estén relacionados entre sí será de forma conjunta.

Ya hemos comentado en varios puntos, que un desarrollo distribuido introduce nuevas fuentes potenciales de fallos, que en este tipo de test nos podrán afectar en gran medida. En concreto tendremos en cuenta que un fallo de este test se puede dar por un mal funcionamiento del software, evidentemente, pero también tendremos que descartar que no se produzca por la arquitectura subyacente que soporta el sistema. En caso de duda del fallo, repetiremos los test en otro momento para comprobar si se repite el fallo o era un fallo de otro tipo de elementos.

Otro de los puntos más determinantes es quién crea los test y cómo se crean. En otro tipo de test los responsables de creación son los propios equipos de desarrollo del ms, en este caso la responsabilidad debería recaer sobre un conjunto de programadores de todos los ms involucrados que conozcan a fondo el problema que se está implementando de forma conjunta, no exclusivamente su ms.

### Implementación de otros tipos de test

**Test de producción**. Cuando desarrollamos una plataforma no tenemos acceso a datos reales de ejecución, con lo que los test que realizamos no son realistas. Para suplir esta carencia, es muy recomendable realizar test en el entorno de producción, una vez desplegado el sistema y funcionando.

Para este tipo de test debemos ser muy cuidadosos ya que el sistema está en funcionamiento y si realizamos algún tipo de acción dañina repercutirá inmediatamente en los clientes, con lo que este tipo de test tienen que ser extremadamente seguros y que no causen ningún tipo de problema, usando la latencia de los servicios como una métrica para determinar el impacto del test en el rendimiento del sistema.

**Test de rendimiento**. Este tipo de test nos aseguran que los requerimientos de rendimiento se cumplen en nuestro sistema. Cuando desarrollamos y probamos un ms de forma unitaria no aparecen problemas de comunicaciones generalmente, con lo que es ahora cuando detectaremos si la latencia de comunicación entre los ms influye en el funcionamiento. Siempre que hagamos este tipo de test intentaremos usar las mismas herramientas y métricas que se implementan en el entorno de desarrollo, para tener los datos más parecidos a los reales que sea posible.

**Test de robustez**. El fallo de un ms en concreto no debería de hacer que nuestro sistema fallara por completo, con lo que este tipo de pruebas van dirigidas a este tipo de problemas. Haremos que uno o varios ms fallen y veremos cómo reacciona todo el conjunto. Evidentemente estamos dando por sentado que son sistemas en el que los ms se ejecutan de forma paralela y hay sistemas de gestión de carga.

## Monitorización

Con el desarrollo de un sistema distribuido basado en ms el número potencial de elementos que pueden fallar crece exponencialmente. Es una tarea imprescindible comprobar en todo momento que nuestro sistema funciona y que funciona correctamente, para tal fin tenemos las tareas de monitorización, encargadas de velar por la estabilidad del sistema. La monitorización que vamos a implementar dependerá del tipo de sistema que hemos desarrollado.

* Un ms por servidor. En este caso recogeremos información del host sobre el estado de los recursos y le añadiremos la información que genere el ms para determinar el estado, atenderemos también al tiempo de respuesta que está teniendo el ms para poder detectar el inicio de un fallo.
* Un ms en varios servidores. En este caso el procedimiento es similar al anterior, pero intentando ejecutar el mismo comando en varios hosts a la vez para recoger en el mismo instante toda la información (multiplexores de ssh), agregando posteriormente toda la información que genere el ms.
* Varios ms en varios servidores. Al existir varios servidores el procedimiento de agregación de datos es fundamental, hay que crear un único archivo de logs a partir de varios con información sobre varios ms.

Como hemos visto, para poder monitorizar bien una aplicación basada en ms es imprescindible tener los logs y tenerlos bien agregados. Eso significa que los datos estén en un formato coherente y que se puedan estudiar fácilmente para determinar los posibles problemas. Para minimizar esfuerzos se pueden seguir las siguientes recomendaciones a la hora de crear y agregar logs:

* Hacer un repositorio central para la gestión de logs con el que se comuniquen los sistemas que alberguen los ms. Esta aproximación introduce un nuevo punto de fallo en la arquitectura.
* Antes de diseñar cualquier aplicación basada en ms estudiar las herramientas de logs existentes en el sistema, e implementar una en nuestra institución, siendo esta la que se usará después por parte de los ms para realizar los logs.
* Establecer un formato estandarizado de logs para todos los host y ms existentes, de tal manera que sea sencillo agregarlos y reconocer la información. En este punto es imprescindible crear IDs únicos para cada transacción que vaya viajando por los ms para poder identificar el fallo y la cadena de ejecución. Por supuesto este ID se deberá recoger en los logs junto con la información de tránsito y de ejecución en caso de fallo.
* Recoger la fecha y hora en la que un evento sucede parece una buena aproximación en principio, pero tendremos que considerar que los relojes de los servidores raramente están sincronizados al microsegundo, que las zonas horarias pueden producir problemas a la hora de agregar y algunos problemas más.
* Por último, una consideración de eficiencia. Todo proceso de log hará que nuestro sistema sea más lento, pues debe mandar información al exterior y almacenarse (en disco o en red) con lo que habrá que equilibrar la cantidad de información y número de sucesos monitorizados con el rendimiento del sistema.

## Seguridad

La seguridad es un tema muy amplio. Ser experto implica un nivel de formación y experiencia que no tenemos, por lo que solamente daremos unas pinceladas y criterios muy básicos a tener en cuenta en este sentido.

Los principios básicos de seguridad se podrían enumerar (y no ser los únicos ni probablemente los más importantes):

* Otorgar el mínimo privilegio posible al usuario o al proceso para realizar su labor, así como ejecutarlo el mínimo tiempo.
* Tener múltiples protecciones para defenderse.
* Realizar diferentes controles de seguridad: preventivos, de detección y de respuesta.
* Utilizar la automatización en la seguridad como primera herramienta, pero no como exclusiva.

La actividad de seguridad informática es muy extensa y dedicada, enumera o intentar hacer un listado exhaustivo de las funciones que tiene un experto en seguridad es impensable. Por ellos, las siguientes funciones nos parecen las más relevantes en cuanto al diseño de ms y la seguridad:

* Identificar los potenciales atacantes.
* Proteger los recursos de los atacantes.
* Detectar los posibles ataques.
* Responder a los ataques de forma adecuada.
* Recuperarse del incidente.

Los fundamentos que tendremos en cuenta para aplicar la seguridad se resumen en:

* Gestión de credenciales. Tanto de usuario como de los ms implicados. En cuanto a los usuarios deben cumplir con los requerimientos de longitud y complejidad, así como del rotado de las claves. En cuanto a los ms y datos podemos usar cualquier esquema de seguridad actual: SSL, TSL, siempre que tengamos en cuenta la creación, distribución, almacenamiento, rotación, monitorización y revocación de las claves.
* Asegurar los datos. En este caso se tendrán en cuenta tanto los datos en tránsito, para que no puedan ser modificados, así como la identidad del cliente y del servidor.
* Gestionar la autorización del usuario de forma correcta, no debemos obligar al usuario a autentificarse cada vez que haga una petición.

Para poder incluir una buena seguridad en nuestros diseños tendremos que incorporar a los equipos multidisciplinares que se crean para el desarrollo una persona de seguridad, que entienda el proyecto y tenga experiencia en programación.

## Escalado

El diseñar una aplicación basada en servicios nos va a proporcionar atender a gran cantidad de peticiones y adaptarnos a los cambios del mercado rápidamente. Pero evidentemente, no vamos a tener al principio las mismas necesidades de tratamiento que después de un tiempo de evolución de nuestro negocio. Por lo que, en la vida de nuestra empresa, es muy probable que tengamos que añadir más servicios, o crear más instancias de los existentes, o dar respuesta a situaciones de crecimiento no esperadas. Esta tarea es la que denominamos escalado.

El escalado es cómo nuestra arquitectura va a crecer para dar respuesta al crecimiento de la empresa. En este sentido hay dos tipos de escalado:

* Vertical. Añadiremos máquinas a nuestro hardware.
* Horizontal. Añadiremos más recursos en paralelo, duplicaremos aquellas partes que lo necesiten, generalmente usando además balanceadores de carga.

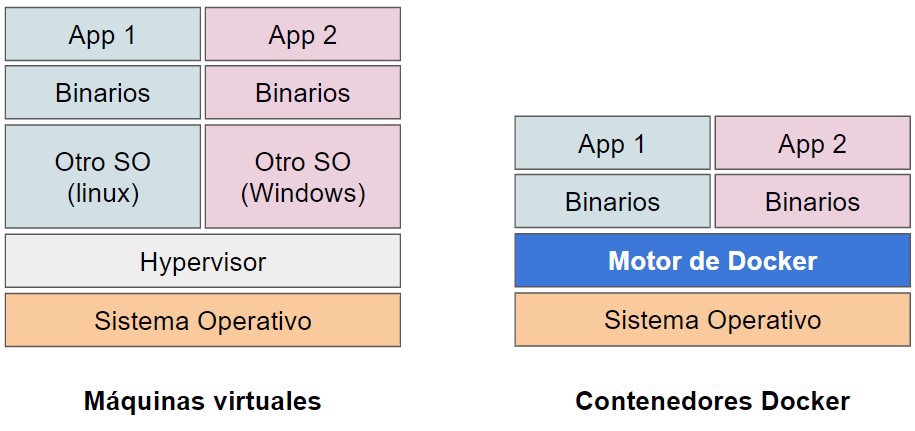
Los tipos de escalado ante los recursos están claros, pero qué pasa si lo que necesitamos es que la gestión de nuestros datos sea más eficiente y soporte más carga. En estos casos se puede añadir a las dos anteriores una tercera aproximación, en la que los datos se dividen en grupos por algún tipo de criterio (por ejemplo, alfabético) y se distribuyen las cargas en función de las peticiones.

A la hora de escalar, tendremos en cuenta siempre el objetivo que queremos llevar a cabo, aumentar recursos por aumentar no tiene sentido, tenemos que definir muy bien la hoja de ruta que vamos a desarrollar. En concreto, escalaremos para aumentar el rendimiento y ser robustos.

El diseñar una buena estrategia de escalado dependerá del diseño inicial que tengamos del sistema. Si este es monolítico no queda más que un enfoque vertical, por el contrario, si tiene elementos constituyentes podremos adoptar algún tipo de mezcla vertical y horizontal, mientras que, si el desarrollo está basado en ms, el enfoque más apropiado a usar será el horizontal.

# Capítulo IV. Uso de Docker

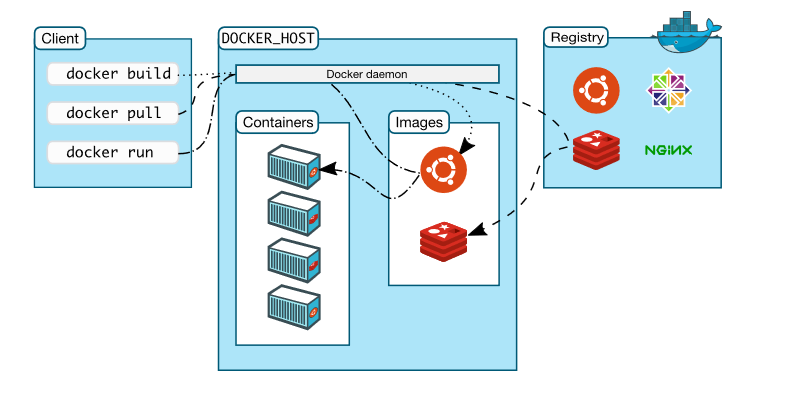
La aparición de Docker ha revolucionado el modo de desplegar y desarrollar aplicaciones. Permite aislar código dentro de un entorno seguro evitando interaccionar con el sistema anfitrión. Es muy parecido a las máquinas virtuales, pero sin la carga de las máquinas virtuales y una flexibilidad para el despliegue que no puede proporcionar una máquina virtual.

Docker permite empaquetar aplicaciones completas en contenedores que incluyen todo lo necesario para su ejecución de forma aislada. En un contenedor Docker existirán las librerías necesarias imprescindibles para la ejecución de la aplicación, la aplicación en sí y el software de control de Docker. Esta configuración es mucho más liviana que si creamos una máquina virtual completa que deberá tener todo el SSOO más los entornos software necesarios para nuestra aplicación.

El concepto principal de Docker es el contenedor, alrededor de él se crea todo un ecosistema de funciones para ejecutar de forma aislada nuestra aplicación. Un contenedor es una unidad de software que permite empaquetar el código de una aplicación con todas sus dependencias y ejecutarla. Lo general es que cada contenedor tenga un único proceso (aplicación) ejecutándose a la vez.

## Arquitectura

La arquitectura Docker recoge todos los elementos del entorno y sus diferentes herramientas, así como las interacciones entre las diferentes herramientas.



* **Cliente**. Se encarga de servir de intermediario entre el usuario y el sistema Docker, muestra comandos para gestionar tanto el host como el registro.
  + **Docker Cli**. Creado en dos versiones, es un interfaz en modo comando (***docker***) que presenta todas las órdenes de gestión. Se utiliza cuando el número de contenedores es pequeño o uno.
  + **Docker Compose**. También es una aplicación en modo comando (*docker-compose* o d*ocker compose*) que permite definir a través de un único fichero de configuración todos los contenedores de nuestra aplicación y gestionarlos. Se utiliza cuando hay más de un contenedor. Además de gestionar todo el ciclo de vida de todos los contenedores definidos, podrá gestionar el reinicio de ellos en caso de caída.
* **Docker Engine (Docker host)**. Es el encargado principal del sistema, realiza las funciones de crear, ejecutar y gestionar los contenedores, se accede a través de un cliente obligatoriamente. El componente principal del Docker Engine es el Docker Daemon (Se ejecuta de forma nativa en Linux y Windows Server, de forma virtualizada en el resto de sistemas Windows y Mac). La arquitectura Docker está basada en cliente – servidor, por lo que cliente y servidor no tienen la necesidad de residir en la misma máquina física utilizando una comunicación TCP bajo el puerto 2375, hay que tener en cuenta que esta comunicación es no cifrada, para entornos de producción usaremos TSL sobre el puerto 2376.
* **Docker Registry**. Es el encargado de almacenar y distribuir imágenes Docker. Hemos dicho que el componente principal es el contenedor. Un contenedor es una aplicación que tiene un ciclo de vida, pero cuando no está ejecutándose esa aplicación reside en un fichero llamado imagen. La imagen se usa para crear el contendor a la hora de la ejecución. Haciendo un símil con la programación orientada a objetos la imagen sería a la clase lo que el contenedor al objeto.

Por tanto, para crear un contenedor hace falta una imagen y esta debe residir en algún sitio. El Docker registry es el lugar físico en dónde se almacenan las imágenes para estar accesibles para la creación de contenedores. Este espacio físico debe residir en el host que ejecutar Docker antes de crear el contenedor. Y tiene una estructura determinada. El registro está formado por repositorios, que son conjunto de versiones de la misma imagen. Cada imagen pertenecerá a un repositorio y tendrá un nombre (etiqueta) para identificarla. Así para acceder a una imagen se hará con la nomenclatura **repositorio:etiqueta**.

Si bien todas las imágenes tienen que estar en local para crear el contenedor, existe un repositorio global desde el que descargar a nuestro registro local imágenes para no tener que crearlas desde cero. Este repositorio se llama Docker Hub. Es el repositorio oficial de imágenes, pero no es el único. Es el que se utilizará por defecto para las descargas de las imágenes en caso de no existir en local.

### Objetos

El sistema Docker utiliza los siguientes objetos para su funcionamiento.

* **Imágenes**. Contienen los sistemas de archivos que utilizarán los contenedores. Son obligatorias para crear los contenedores. A a partir de una imagen se pueden crear tantos contenedores como necesitemos.
* **Contenedores**. Se crean a partir de la imagen. Son un proceso aislado de todos los demás que corren en la máquina host. Existen contenedores para Linux y contenedores para Windows. En estos apuntes nos centraremos en los primeros al ser más eficientes.
* **Volúmenes**. Es el mecanismo que tiene Docker para asegurar la persistencia de los datos. Cuando un contenedor se para, todas sus estructuras desaparecen de memoria, y si no usamos volúmenes, los datos que se hayan generado o almacenado dentro del mismo se borrarán. Un contenedor tendrá asociados uno o varios volúmenes, así como un volumen podrá estar asociado a uno o varios contenedores. El ciclo de vida de un contenedor es independiente del ciclo de vida del contenedor.
* **Redes**. Los contendores nacen sobre todo para solventar problemas en red, por lo que Docker permite una gestión avanzada de redes para sus contenedores. Se pueden crear diversas redes y de diferentes tipos, aisladas o no del exterior.

Los objetos mencionados anteriormente los iremos desgranando en puntos posteriores de este capítulo.

Y Para terminar la arquitectura Docker, faltaría un elemento que no es propiamente de la arquitectura pero que se presenta imprescindible cuando queremos desplegar las aplicaciones Docker en clusters de redes para aumentar la disponibilidad. Son los “orquestadores” de contenedores. Dentro de esta categoría aparece Kubernets y Docker Swarm. El último viene integrado con el propio sistema Docker, pero el más utilizado hoy en día es Kubernets que será tratado en un capítulo posterior.

## Instalación de Docker

**$ lsb\_release -a**

No LSB modules are available.

Distributor ID: Ubuntu

Description: Ubuntu 21.10

Release: 21.10

Codename: impish

**$ docker**

No se ha encontrado la orden «docker», pero se puede instalar con:

sudo snap install docker # version 20.10.12, or

sudo apt install docker.io # version 20.10.7-0ubuntu5.1

Consulte «snap info docker» para ver más versiones.

**$ sudo apt install docker.io**

**$ docker –versión**

Docker version 20.10.7, build 20.10.7-0ubuntu5.1

**$ sudo systemctl status docker**

● docker.service - Docker Application Container Engine

Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)

Active: active (running) since Mon 2022-04-11 11:44:11 CEST; 1min 52s ago

TriggeredBy: ● docker.socket

Docs: https://docs.docker.com

Main PID: 1780 (dockerd)

Tasks: 10

Memory: 41.6M

CPU: 154ms

CGroup: /system.slice/docker.service

└─1780 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Vamos a probar la instalación.

**$ sudo docker run hello-world**

Unable to find image 'hello-world:latest' locally

latest: Pulling from library/hello-world

2db29710123e: Pull complete

Digest: sha256:bfea6278a0a267fad2634554f4f0c6f31981eea41c553fdf5a83e95a41d40c38

Status: Downloaded newer image for hello-world:latest

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.

2. The Docker daemon pulled the "hello-world" image from the Docker Hub.

(amd64)

3. The Docker daemon created a new container from that image which runs the

executable that produces the output you are currently reading.

4. The Docker daemon streamed that output to the Docker client, which sent it

to your terminal.

…

**$ sudo docker images**

REPOSITORY TAG IMAGE ID CREATED SIZE

hello-world latest feb5d9fea6a5 6 months ago 13.3kB

**$ sudo docker version**

Client:

Version: 20.10.7

API version: 1.41

Go version: go1.13.8

Git commit: 20.10.7-0ubuntu5.1

Built: Mon Nov 1 00:33:40 2021

OS/Arch: linux/amd64

Context: default

Experimental: true

Server:

Engine:

Version: 20.10.7

API version: 1.41 (minimum version 1.12)

Go version: go1.13.8

Git commit: 20.10.7-0ubuntu5.1

Built: Thu Oct 21 23:58:58 2021

OS/Arch: linux/amd64

Experimental: false

containerd:

Version: 1.5.5-0ubuntu3.1

GitCommit:

runc:

Version: 1.0.1-0ubuntu2

GitCommit:

docker-init:

Version: 0.19.0

GitCommit:

Por último, configuramos el usuario no privilegiado para que use el entorno sin tener que validarse con la orden **sudo** en el sistema, añadiéndolo al grupo docker.

**$ sudo usermod -aG docker $USER**

-- Salir de la sesión y volver a entrar --

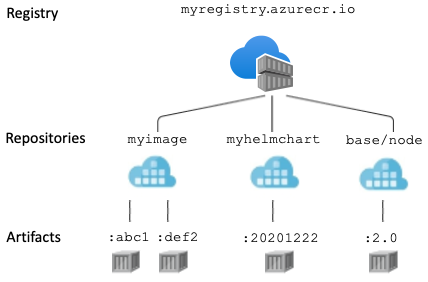
**$ groups**

usuario adm cdrom sudo dip plugdev lpadmin lxd sambashare **docker**

## Imágenes Docker

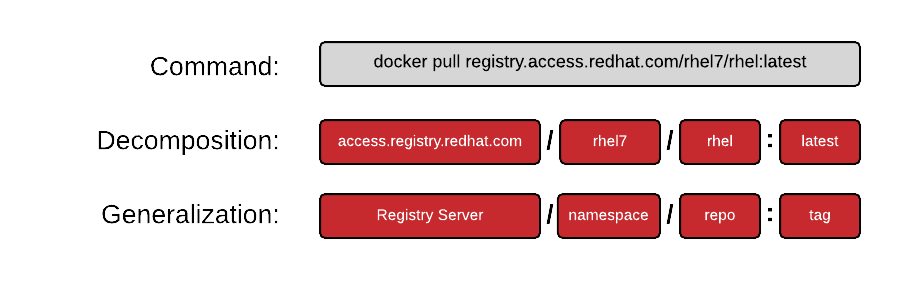
Una imagen es una plantilla de solo lectura que usamos para crear los contenedores. Estas plantillas incluyen todos los ficheros necesarios para la ejecución de la aplicación en un sistema de ficheros propios, sobre esta imagen se aplica un algoritmo hash para crear un identificador único, este identificador se llama digest.

Las imágenes están formadas por capas. Cada capa incluye las modificaciones con la capa anterior, pero no elimina nada, con lo que las operaciones de borrado sobre la creación de la imagen nunca reducirán el tamaño de la misma. Las capas se crean con cada instrucción que se ejecutan al crear la imagen.



Ya hemos mencionado que las imágenes deben residir de forma local para poder crear los contenedores. Las imágenes se organizan en repositorios tanto en local como en los servidores compartidos en red. Dentro de un repositorio residen todas las imágenes que deseemos y las identificaremos por un nombre propio (etiqueta o tag) para distinguir unas de otras. Si bien no es obligatorio que las imágenes de un repositorio estén relacionadas, esto debería ser así, debiendo estar en el mismo repositorio las diferentes versiones de la misma imagen, siendo la etiqueta la versión de la imagen.

Del mismo modo que las imágenes se agrupan en repositorios, los repositorios se agrupan en entidades de orden superior llamadas espacios de nombre. Estos espacios de nombre se usan para organizar los repositorios dentro de un registro de contenedores.



Otro ejemplo con un contenedor de python3:

**$ docker run python:slim-buster**

unable to find image 'python:slim-buster' locally

slim-buster: Pulling from library/python

f003217c5aae: Pull complete

89f192ed0478: Pull complete

f7cb0a75e286: Pull complete

2040e058b101: Pull complete

343df816b959: Pull complete

Digest: sha256:48c4a798972f30b485fea59076ec186644da63c1845f38e32b6c2fada51e7144

Status: Downloaded newer image for python:slim-buster

**$ docker run -it python:slim-buster**

Python 3.10.4 (main, Apr 7 2022, 03:47:55) [GCC 8.3.0] on linux

Type "help", "copyright", "credits" or "license" for more information.

**>>> print("Hello")**

Hello

**>>> import sys**

**>>> sys.version**

'3.10.4 (main, Apr 7 2022, 03:47:55) [GCC 8.3.0]'

**>>> exit()**

### El archivo Dockerfile

El archivo Dockerfile es un archivo de texto que describe las instrucciones precisas para crear una imagen a partir de otra imagen. Este último punto es muy importante, siempre partiremos de una imagen base que modificaremos para nuestras necesidades. Como esta imagen que estamos creando se ejecutará en memoria intentaremos utilizar como base la imagen más pequeña que podamos, para reducir el tamaño final.

El fichero está formado por instrucciones, estas instrucciones deben escribirse en mayúsculas (es una recomendación) y se ejecutarán de arriba🡪abajo, ignorando todas aquellas líneas que comiencen por una almohadilla (#).

Veamos el Dockerfile con el que se crea la imagen hello-world (https://hub.docker.com/\_/hello-world, latest):

**FROM** scratch

**COPY** hello /

**CMD** ["/hello"]

En el fichero aparecen tres órdenes (FROM, COPY y CMD) con sus correspondientes parámetros. En esta sección vamos a ver todas las órdenes posibles para este fichero, así como sus relaciones. La creación del fichero Dockerfile es la base de los contenedores y uno de los puntos críticos del problema. Estamos creando con este fichero la plantilla de la imagen que posteriormente se ejecutará en memoria.

#### Variables (ARG)

Podemos crear variables en nuestro fichero Dockerfile para usarlas a lo largo de todo él. La creación se hace con el comando ARG, definiendo un nombre de variable y un valor (**ARG VARIABLE=valor**). Posteriormente haremos uso de la variable mediante la sintaxis **${NOMBRE\_VARIABLE}.** Esta variable solo estará disponible en el fichero Dockerfile, nunca dentro de la imagen. Estas variables se pueden sobrescribir durante el periodo de creación usando el parámetro **--builds-arg**.

**ARG** VERSION=3.7

**FROM** python:{$VERSION}

**CMD** ["/hello"]

#### IMAGEN BASE (FROM)

Como se ha indicado es imprescindible seleccionar una imagen base para modificarla, en caso que queramos empezar desde cero usaremos **scratch.** Esta instrucción se puede utilizar tantas veces como deseemos en el fichero, pero lo general es que sea solo una. En caso que se use varias veces es para reducción del tamaño de la imagen final. Lo que hay que considerar que un fichero Dockfile solo creará una imagen, independientemente del número de imágenes intermedias que hayamos creado nosotros.

La sintaxis de la instrucción es **FROM repositorio:etiqueta AS alias**. Si omitimos la etiqueta se tomará por defecto **latest**. La parte final define un alias opcional que se suele utilizar en los Dockerfiles de varias etapas.

#### Variables de entorno (ENV)

Al contrario que las variables de fichero, estas variables estarán disponibles dentro de la imagen como variables de entorno. Dependiendo el sistema operativo de la imagen que estemos creando serán accesibles de una manera u otra. Así en entornos Linux podemos usar **printenv** para comprobar todas las variables existentes. Se pueden sobrescribir al ejecutar un contenedor mediante el argumento **-env¸** o cargar desde un fichero con **--env-file**.

**ARG** VERSION=3.7

**ARG** PATH\_LOCAL=/var/www

**FROM** python:{$VERSION}

**ENV** PATH=$PATH\_LOCAL:$PATH

**CMD** ["/hello"]

En este ejemplo creamos dos variables de fichero (VERSION, PATH\_LOCVAL), pero prestaremos atención ahora a la segunda: **PATH\_LOCAL**, que la vamos a usar para modificar la variable de entorno PATH. Esta variable se modifica añadiendo por delante nuestra variable de fichero y a continuación añade lo que la propia imagen haya configurado al crearla. En el momento que creemos un contenedor a partir de esta imagen, internamente la variable PATH contendrá /var/www:resto\_de\_path y será accesible para cualquier aplicación.

Este mecanismo se usa constantemente para los parámetros de configuración de nuestros contenedores. Así, por ejemplo, si creamos un contendor de mysql, los usuarios y claves se pasarán a través de variables de entorno, así como la base de datos a utilizar por defecto.

#### Ejecución de comandos dentro de la imagen (RUN)

Una vez hemos configurado las variables y definido nuestra imagen base, tendremos que modificarla para nuestros deseos. Generalmente estas modificaciones se hacen ejecutando comandos dentro del SSOO de la imagen que estamos creando. Por tanto, estas instrucciones son dependientes del SSOO de la imagen que utilicemos. Nosotros nos vamos a ceñir a usar imágenes Linux, pero aun así no es lo mismo usar una distribución Debian/Ubuntu que una RedHat, por lo que los comandos tienen que adecuarse a la distribución que usemos.

Cada vez que usamos una instrucción RUN se crea una nueva capa en la imagen. Tenemos que recordar que cada capa contiene las modificaciones con la anterior y que incluso si damos órdenes de borrado, el tamaño de la imagen no cambiará, solo se reflejarán los cambios. Por tanto, el diseño de las órdenes RUN debe ser meticuloso, intentando realizar el mayor número de comandos simultáneos en la misma orden siempre que estén relacionados. De esta manera crearemos capas lo más reutilizables posible y los posteriores procesos de creación se acelerarán. Para poder realizar varios comandos en una única línea (bajo Linux) será **&&** para unir comandos y **\** al final de cada línea que queramos partir.

**RUN** set -eux \

&& savedAptMark="$(apt-mark showmanual)" \

&& apt-get update \

&& apt-get install -y --no-install-recommends \

dpkg-dev \

gcc

Existe una sintáxis alternativa en la que los **&&** se sustituyen por **;.**

**RUN** set -eux; \

savedAptMark="$(apt-mark showmanual)"; \

apt-get update; \

apt-get install -y --no-install-recommends \

dpkg-dev \

gcc;

#### Copiar ficheros dentro de la imagen (copy)

Muchas veces necesitaremos copiar a la imagen ficheros propios tales como: ficheros de configuración, ficheros de la aplicación, scripts de inicialización, etc. En estos casos usamos el comando COPY. La sintaxis es **COPY [--chown=user:group] <origen> <destino>**. Tanto origen como destino pueden ser un directorio o un fichero y se copiarán a la máquina con el usuario y grupo por defecto que será root:root si no lo hemos cambiado con USER o no hemos especificado la parte opcional --chown.

**COPY** src /app

**COPY** file.conf /var/log

**COPY** –chown=www-data:www-data /html /var/www

En los ejemplos anteriores vemos las tres posibilidades, el primero directorio a directorio, el segundo fichero a directorio y el tercero directorio a directorio con cambio de propietario y grupo.

#### Puertos de la imagen (EXPOSe)

El sistema Docker se utiliza generalmente para crear entornos distribuidos, con lo que es muy normal que los procesos que se ejecuten sean servidores que deben accederse a través de puertos TCP/UDP. Esta instrucción indica qué puertos utiliza la imagen, pero no los hace públicos, es necesario posteriormente en la creación de contenedor (*run -p*) especificarlos.

**EXPOSE** 80 tcp/3000

#### Almacenamiento persistente (VOLUME)

Todo contenedor que se ejecuta tiene todas sus capas de solo lectura excepto la última que es de lectura – escritura. Pero esta capa se elimina en el momento que el contenedor deja de ejecutarse, perdiéndose todos los datos o modificaciones que se hayan hecho en ejecución.

Para evitar la pérdida de datos se utilizan los volúmenes. El volumen tiene un ciclo de vida diferente al del contenedor, pudiendo ser compartido además por varios de ellos.

Veremos esta facilidad con mucho más detenimiento en un punto posterior.

#### Cambio del directorio por defecto (workdir)

El directorio por defecto al crear la imagen es la raíz del sistema de ficheros. Esto puede no ser conveniente a la hora de ejecutar scripts o de hacer copias masivas. Podemos definir el directorio base tantas veces como necesitemos y las subsiguientes instrucciones tomarán de referencia dicho directorio en vez de la raíz.

**WORKDIR** /var/www

**COPY** src .

Copiamos el directorio src en el directorio /var/www de la imagen, establecido con WORKDIR.

#### Cambio del USUARIO por defecto (USER)

Ya hemos comentado en el comando de copia que los ficheros se copiarán con el usuario root y el grupo root si no lo cambiamos y vimos una forma de hacerlo dentro del mismo comando. Esta forma cambia el usuario por defecto con el que se ejecutarán las instrucciones (RUN, CMD y ENTRYPOINT) a partir de este momento. Tiene las mismas implicaciones que si en una consola Linux estamos trabajando con ese usuario en vez de con el root.

**USER** usuario:users

**RUN** myapp.py

A partir de este momento las instrucciones se ejecutarán como bajo *usuario* en vez de *root*, con lo que la instrucción COPY se establecerá al propietario usuario con sus permisos por defecto y al grupo users.

#### Comando de inicio (ENTRYPOINT - CMD)

ENTRYPOINT define el primer comando que se ejecutará cuando se cree el contenedor (run). Solo puede aparecer una única vez o puede omitirse, en este caso el comando aparecerá obligatoriamente en la instrucción CMD.

Si se define este comando, la instrucción CMD tendrá los parámetros necesarios, si no se define este parámetro, CMD deberá tener todo el comando completo.

**ARG** VERSION=3.7

**ARG** PATH\_LOCAL=/var/www

**FROM** python:{$VERSION}

**ENV** PATH=$PATH\_LOCAL:$PATH

**CMD** ["cat",“/etc/os-release”]

**ARG** VERSION=3.7

**ARG** PATH\_LOCAL=/var/www

**FROM** python:{$VERSION}

**ENV** PATH=$PATH\_LOCAL:$PATH

**ENTRYPOINT** [“cat”]

**CMD** [“/etc/os-release”] # solo parámetros

Los valores de CMD y ENTRYPOINT puede ser una lista como vemos en los ejemplos o el texto completo de la orden tal y como se ejecutaría en el Shell.

#### Referencia

Hemos repasado los comandos más importantes, para una lista y descripción más detallada hay que visitar:

https://docs.docker.com/engine/reference/builder/

### Creación de imágenes

La creación de contenedores se basa en instanciar imágenes como procesos y ejecutarlas en memoria. Las imágenes son el punto central del sistema, siendo el soporte de toda operación bajo Docker. Existen imágenes en los repositorios que simplemente tenemos que descargar, pero por lo general necesitaremos crear la nuestra propia.

La creación de una imagen tiene los siguientes pasos:

1. Creación del fichero Dockerfile con las instrucciones necesarias.
2. Generación de la imagen mediante la orden: *docker build parámetros*. En este ejemplo solo usaremos el parámetro t para indicar la etiqueta con la que vamos a crear la imagen, como solo tiene una parte, añadirá **latest** de forma automática.
3. Comprobación que la imagen se ha cargado en el repositorio local: *docker images*.
4. Creación y ejecución de un contenedor: *docker run nombre\_imagen.*

#### Ejemplo de creación

#### 

**main.py**

print(“Hola Mundo”)

**Dockerfile**

FROM python:slim-buster

WORKDIR /app

COPY main.py .

CMD [“python”,”main.py”]

**$docker image build -t primera-imagen .**

4 : FROM python:slim-buster

---> 1d6f4434a01b

Step 2/4 : WORKDIR /app

---> Running in 33a5e65f3602

Removing intermediate container 33a5e65f3602

---> 111caba81577

Step 3/4 : COPY main.py .

---> 8ac61cb07032

Step 4/4 : CMD [“python”,”main.py”]

---> Running in c0ca35e22db8

Removing intermediate container c0ca35e22db8

---> b3d9bf857a43

Successfully built b3d9bf857a43

Successfully tagged primera-imagen:latest

**$docker image ls**

REPOSITORY TAG IMAGE ID CREATED SIZE

**primera-imagen latest b3d9bf857a43 30 seconds ago 118MB**

python slim-buster 1d6f4434a01b 12 days ago 118MB

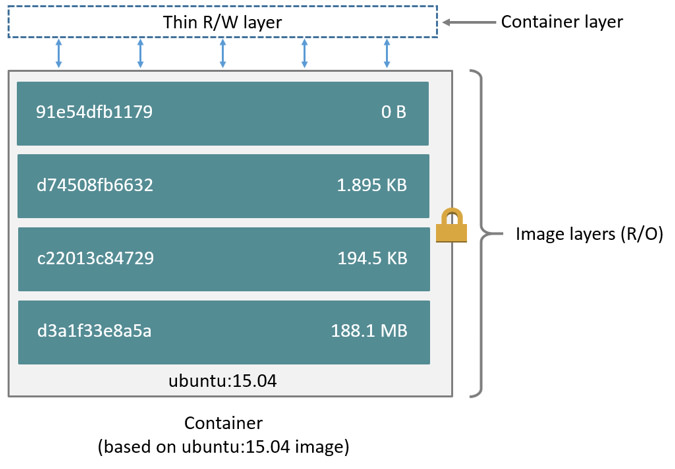
hello-world latest feb5d9fea6a5 6 months ago 13.3kB

**$docker container run primera-imagen**

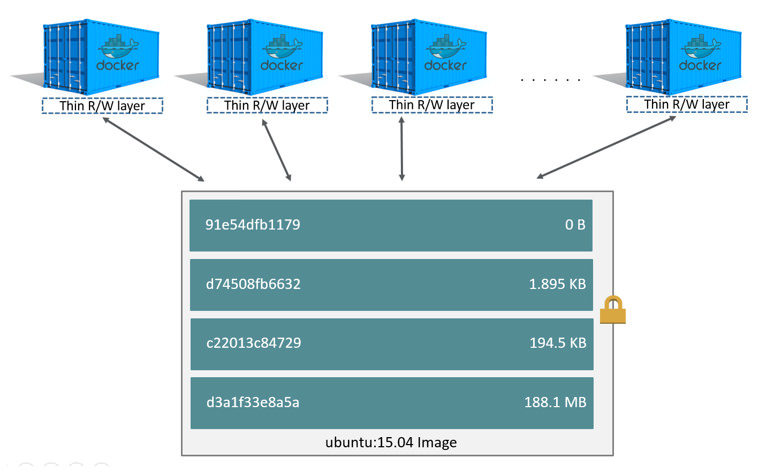
Hola Mundo

### Capas de una imagen

Toda imagen de Docker está formada por la unión de múltiples capas que contienen las modificaciones que se han realizado sobre la capa anterior o la imagen de base en el primer paso. Cada capa se corresponde con una instrucción del archivo Dockerfile y se crean de solo lectura con lo que no se pueden modificar una vez hecha la imagen. Todas las instrucciones generan una capa, pero solo las instrucciones COPY, RUN y ADD las que aparecerán en la imagen final, el resto se eliminarán antes de la creación final.



Para los contenedores, al instanciarla de una imagen, se añade en la parte final una nueva capa de lectura – escritura que será desechada en el momento que se termine el contenedor, con lo que los datos de la mismas desaparecerán, el resto de capas permanecerán inmutables y serán compartidas entre todos los contenedores que se ejecuten a partir de la misma imagen.



Uno de los elementos que podemos controlar es cómo se mezclan los archivos de dichas capas, para esta gestión existen diferentes drivers con funcionalidades diferentes, que quedan fuera de este libro, se puede ampliar esta información en la documentación oficial.

https://docs.docker.com/storage/storagedriver/select-storage-driver/

La historia de una imagen nos permite ver las capas y sus tamaños en el momento de la creación de la imagen, incluyendo las de la imagen de base.

**$docker image history primera-imagen**

d2cc5bdf9820 21 minutes ago /bin/sh -c #(nop) CMD ["python" "main.py"] 0B

225a682b34e6 23 minutes ago /bin/sh -c #(nop) COPY file:33dc8f38cbb43646… 21B

111caba81577 31 minutes ago /bin/sh -c #(nop) WORKDIR /app 0B

1d6f4434a01b 12 days ago /bin/sh -c #(nop) CMD ["python3"] 0B

<missing> 12 days ago /bin/sh -c set -eux; savedAptMark="$(apt-m… 11.3MB

<missing> 12 days ago /bin/sh -c #(nop) ENV PYTHON\_GET\_PIP\_SHA256… 0B

<missing> 12 days ago /bin/sh -c #(nop) ENV PYTHON\_GET\_PIP\_URL=ht… 0B

<missing> 12 days ago /bin/sh -c #(nop) ENV PYTHON\_SETUPTOOLS\_VER… 0B

<missing> 12 days ago /bin/sh -c #(nop) ENV PYTHON\_PIP\_VERSION=22… 0B

<missing> 12 days ago /bin/sh -c set -eux; for src in idle3 pydoc… 92B

<missing> 12 days ago /bin/sh -c set -eux; savedAptMark="$(apt-m… 30MB

<missing> 3 weeks ago /bin/sh -c #(nop) ENV PYTHON\_VERSION=3.10.4 0B

<missing> 3 weeks ago /bin/sh -c #(nop) ENV GPG\_KEY=A035C8C19219B… 0B

<missing> 3 weeks ago /bin/sh -c set -eux; apt-get update; apt-g… 7.08MB

<missing> 3 weeks ago /bin/sh -c #(nop) ENV LANG=C.UTF-8 0B

<missing> 3 weeks ago /bin/sh -c #(nop) ENV PATH=/usr/local/bin:/… 0B

<missing> 3 weeks ago /bin/sh -c #(nop) CMD ["bash"] 0B

<missing> 3 weeks ago /bin/sh -c #(nop) ADD file:59187422476c57db4… 69.3MB

Por otra parte, podemos ver toda la información de bajo nivel de una imagen mediante *inspect*.

**$docker image inspect hello-world**

{

"Id": "sha256:feb5d9fea6a5e9606aa995e879d862b825965ba48de054caab5ef356dc6b3412",

"RepoTags": [

"hello-world:latest"

],

"RepoDigests": [

"hello-world@sha256:bfea6278a0a267fad2634554f4f0c6f31981eea41c553fdf5a83e95a41d40c38"

],

"Parent": "",

"Comment": "",

"Created": "2021-09-23T23:47:57.442225064Z",

"Container": "8746661ca3c2f215da94e6d3f7dfdcafaff5ec0b21c9aff6af3dc379a82fbc72",

"ContainerConfig": {

"Hostname": "8746661ca3c2",

"Domainname": "",

"User": "",

"AttachStdin": false,

"AttachStdout": false,

"AttachStderr": false,

"Tty": false,

"OpenStdin": false,

"StdinOnce": false,

"Env": [

"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"

],

"Cmd": [

"/bin/sh",

"-c",

"#(nop) ",

"CMD [\"/hello\"]"

],

"Image": "sha256:b9935d4e8431fb1a7f0989304ec86b3329a99a25f5efdc7f09f3f8c41434ca6d",

"Volumes": null,

"WorkingDir": "",

"Entrypoint": null,

"OnBuild": null,

"Labels": {}

},

"DockerVersion": "20.10.7",

"Author": "",

"Config": {

"Hostname": "",

"Domainname": "",

"User": "",

"AttachStdin": false,

"AttachStdout": false,

"AttachStderr": false,

"Tty": false,

"OpenStdin": false,

"StdinOnce": false,

"Env": [

"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"

],

"Cmd": [

"/hello"

],

"Image": "sha256:b9935d4e8431fb1a7f0989304ec86b3329a99a25f5efdc7f09f3f8c41434ca6d",

"Volumes": null,

"WorkingDir": "",

"Entrypoint": null,

"OnBuild": null,

"Labels": null

},

"Architecture": "amd64",

"Os": "linux",

"Size": 13256,

"VirtualSize": 13256,

"GraphDriver": {

"Data": {

"MergedDir": "/var/lib/docker/overlay2/af5cd193f982af23483722e1199bb7d263afe044cc292d232b4183d4c4163572/merged",

"UpperDir": "/var/lib/docker/overlay2/af5cd193f982af23483722e1199bb7d263afe044cc292d232b4183d4c4163572/diff",

"WorkDir": "/var/lib/docker/overlay2/af5cd193f982af23483722e1199bb7d263afe044cc292d232b4183d4c4163572/work"

},

"Name": "overlay2"

},

"RootFS": {

"Type": "layers",

"Layers": [

"sha256:e07ee1baac5fae6a26f30cabfe54a36d3402f96afda318fe0a96cec4ca393359"

]

},

"Metadata": {

"LastTagTime": "0001-01-01T00:00:00Z"

}

}

]

### Estructura de las órdenes Docker

Las instrucciones bajo Docker tienen dos sintaxis diferentes. Se recomienda usar la nueva versión ya que la estructura es más coherente e intuitiva con la forma **docker OBJETO ORDEN.**

Los objetos que se deben usar son los que Docker utiliza: image, container, volumen y network. La orden depende del objeto y podemos ver todas las posibles opciones sin no ponemos ninguna orden

### Gestión de las imágenes: docker image

**$docker image**

Usage: docker image COMMAND

Manage images

Commands:

build Build an image from a Dockerfile

history Show the history of an image

import Import the contents from a tarball to create a filesystem image

inspect Display detailed information on one or more images

load Load an image from a tar archive or STDIN

ls List images

prune Remove unused images

pull Pull an image or a repository from a registry

push Push an image or a repository to a registry

rm Remove one or more images

save Save one or more images to a tar archive (streamed to STDOUT by default)

tag Create a tag TARGET\_IMAGE that refers to SOURCE\_IMAGE

Run 'docker image COMMAND --help' for more information on a command.

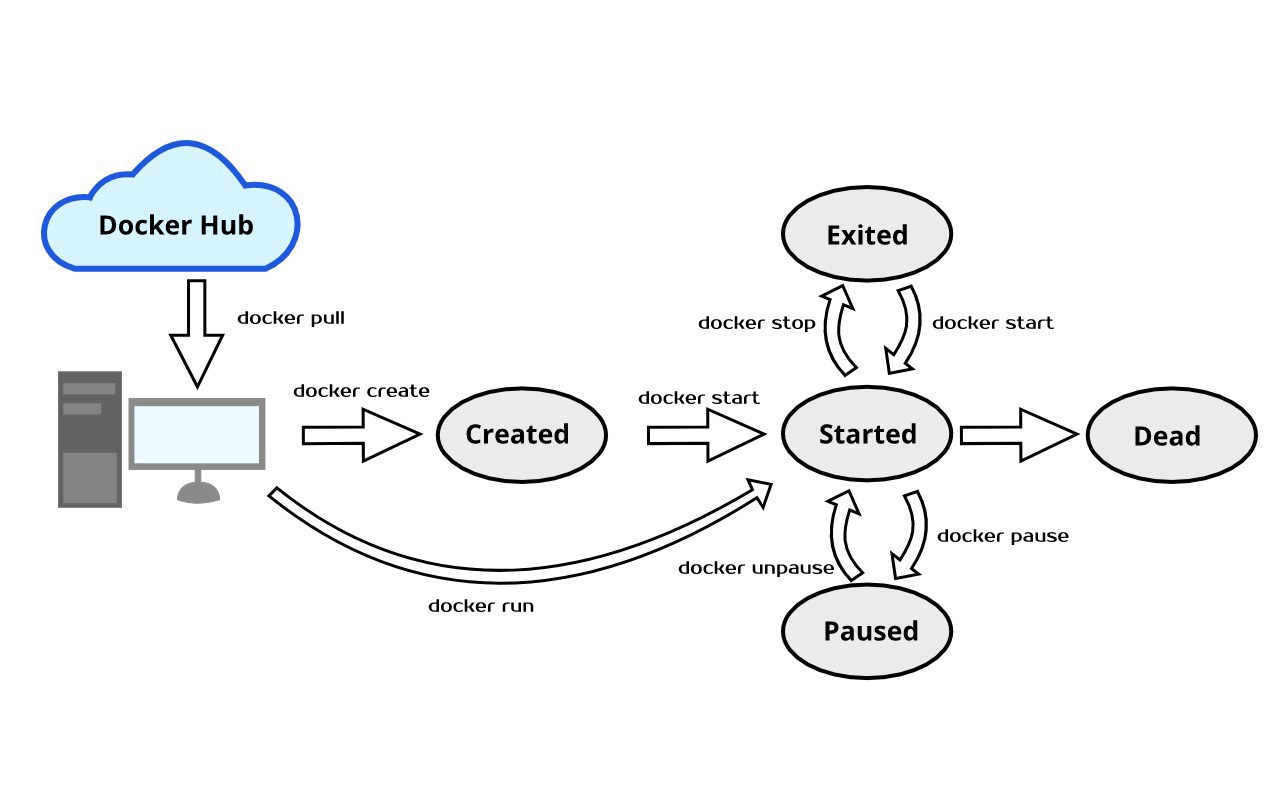
|  |  |  |
| --- | --- | --- |
| Orden | Descripción | Parámetros |
| docker image ls | Muestra el listado de todas las imágenes en el repositorio. | -a Muestra incluso las capas ocultas  -f Permite aplicar filtros de selección para mostrar las imágenes  -q Solo muestra el identificador de la imagen |
| docker image rm imagen | Borra del repositorio local una o varias imágenes. | -f Fuerza el borrado incluso si está en ejecución en algún contenedor |
| docker image prune | Borra todas las imágenes intermedias no usadas que no pertenezcan a ninguna imagen. | -a Borra todas las imágenes no usadas, no solo las que no pertenecen a ninguna imagen.  -f No pide confirmación |
| docker image history | Muestra la historia de la creación de una imagen |  |
| docker image inspect | Muestra información detallada de la imagen |  |

## Contenedores

### Ciclo de vida de un contenedor

Un contenedor es un proceso (generalmente, aunque pueden ser más) que se está ejecutando en una máquina a través del sistema Docker aislado del resto de procesos del sistema operativo de base, con su propio sistema de archivos, interfaz de red y su propio árbol de procesos independiente.

Como todo proceso, un contenedor está en algún estado según la siguiente figura:



* **Created.** El contenedor se ha creado y cargado en memoria, pero no se está ejecutando.
* **Started.** El contenedor está activo, puede realizar la función para la que se ha creado.
* **Paused.** El contenedor está en memoria y preparado, pero no puede recibir ninguna actividad, está parado, en espera de volver a estar en ejecución.
* **Exited.** El contendor ha sido terminado, sigue en memoria, pero todavía no ha sido descargado, no puede aceptar ninguna petición, está finalizando.
* **Dead.** El contenedor está completamente finalizado, permanece en memoria. Este estado suele ser muy breve.

### Gestión de los contenedores: docker container

**$docker container**

Usage: docker container COMMAND

Manage containers

Commands:

attach Attach local standard input, output, and error streams to a running container

commit Create a new image from a container's changes

cp Copy files/folders between a container and the local filesystem

create Create a new container

diff Inspect changes to files or directories on a container's filesystem

exec Run a command in a running container

export Export a container's filesystem as a tar archive

inspect Display detailed information on one or more containers

kill Kill one or more running containers

logs Fetch the logs of a container

ls List containers

pause Pause all processes within one or more containers

port List port mappings or a specific mapping for the container

prune Remove all stopped containers

rename Rename a container

restart Restart one or more containers

rm Remove one or more containers

run Run a command in a new container

start Start one or more stopped containers

stats Display a live stream of container(s) resource usage statistics

stop Stop one or more running containers

top Display the running processes of a container

unpause Unpause all processes within one or more containers

update Update configuration of one or more containers

wait Block until one or more containers stop, then print their exit codes

Run 'docker container COMMAND --help' for more information on a command.

### Creación y ejecución de contenedores: docker container run

Para crear un contenedor témenos que usar la orden *create* y posteriormente ejecutarlo con la orden *start*. Para evitar estos dos pasos, se usa directamente *run* que crea y ejecuta el contenedor de una vez.

El proceso es el siguiente: Se crea una nueva capa de lectura – escritura para el contenedor, se añade a la parte superior de la imagen, a continuación, se crea el contenedor en memoria y se ejecuta el comando que hayamos configurado en el Dockerfile o el que se pase en la orden.

Este es uno de los comandos más complejos y con más opciones, por lo que detallaremos funcionamiento de forma más detenida en los apartados siguientes. En este momento lo que tenemos que saber es que la forma del comando es la siguiente:

**docker container run** PARAMETROS **nombre\_imagen** COMADO ARGUMENTOS

**$docker container run primera-imagen**

Hola Mundo

|  |  |  |
| --- | --- | --- |
| Orden | Parámetro | Descripción |
| docker container run | -d | Ejecuta el contenedor en segundo plano, desligado de una terminal de comandos física. Es el modo de ejecución tradicional para los servicios. |
|  | --entrypoint cmd | Sobrescribe el comando ENDTRYPOINT del Dockerfile |
|  | -e var=valor | Se utiliza para crear una lista de variables de entorno presentes dentro del contenedor durante su ejecución. |
|  | --env-file file | Similar al anterior pero las variables se leerán de un fichero en vez de ponerlas en la orden directamente. |
|  | -i | Hace que el contenedor se ejecute en modo interactivo, pudiendo interactuar a través de la consola con el contenedor. |
|  | --name nombre | Permite dar un nombre al contenedor, si no se asigna se creará uno de forma aleatoria. |
|  | --network nombre | Une el contenedor a una red propia en vez de la red por defecto. |
|  | -p puerto:puerto | Establecer la conexión entre un puerto externo (host) y el interno del contenedor. |
|  | --rm | Descarga el contenedor de memoria una vez finalizado. |
|  | -t | Asigna una termina host al contenedor |
|  | -u uid | Usuario con el que ejecutar la orden, sobrescribe USER del Dockerfile |
|  | -v nombre | Asigna un volumen de almacenamiento al contenedor actual. |
|  | -w dir | Sobrescribe el último WORKDIR del Dockerfile |

#### Ejemplos

**$docker container run --rm -d primera-imagen**

9d17d546b2ef307dbe39d717b70f8e305db982c25edf9481d2b7b6189730008f

**$docker container run -it primera-imagen**

Hola Mundo

**$docker container run --rm -d -p 80:8080 primera-imagen**

055052b3d260d53ced83e7c42e2a53ca435d6f7131cc2cc022e6609fc5122a7a

**$docker container run --rm -d -p 80:8080 –env-file mifile.env primera-imagen**

9d17d546b2ef307dbe39d717b70f8e3ca435d6f7131cc2cc022e6609fc5122a7a

**$docker container run -it primera-imagen bash**

root@1d5649c85450:/app# ls

main.py

root@1d5649c85450:/app#

**$docker container run -it -u 100 primera-imagen bash**

\_apt@78ad56b4e93d:/app$ ls

main.py

\_apt@78ad56b4e93d:/app$

**$docker container run -it -u 100 --name hola\_mundo primera-imagen bash**

**$docker container ls**

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

1af0bf0ef4ef primera-imagen "bash" 9 seconds ago Up 8 seconds **hola\_mundo**

### Listado de contenedores en ejecución: docker container ls

**$docker container ls**

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

108f91112bd4 primera-imagen "bash" 12 seconds ago Up 11 seconds romantic\_shannon

|  |  |  |
| --- | --- | --- |
| Orden | Parámetro | Descripción |
| docker container ls | -a | Muestra todos los contenedores en ejecución posible, independientemente de su estado |
|  | -f | Permite filtrar los contenedores a mostrar. |
|  | -n número | Muestra los últimos número contenedores creados |
|  | -l | Muestra el último contenedor creado |
|  | --no-trunc | Muestra el identificador completo. |

### Usos comunes con los contenedores

#### Ejecutar un shell interactivo (-it)

**$docker container run -it primera-imagen bash**

root@1d5649c85450:/app# ls

main.py

root@1d5649c85450:/app#

#### Borrar de memoria un contenedor al terminar (--rm)

**$docker container run --rm -d primera-imagen**

9d17d546b2ef307dbe39d717b70f8e305db982c25edf9481d2b7b6189730008f

#### Dar un nombre al contenedor (--name)

**$docker container run -it -u 100 --name hola\_mundo primera-imagen bash**

**$docker container ls**

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

1af0bf0ef4ef primera-imagen "bash" 9 seconds ago Up 8 seconds **hola\_mundo**

#### Eliminar uN contenedor incluso si se está ejecutando (-f)

**$docker container run -it -u 100 --name hola\_mundo primera-imagen bash**

**$docker container ls**

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

1af0bf0ef4ef primera-imagen "bash" 9 seconds ago Up 8 seconds hola\_mundo

**$docker container rm -f 1af0bf0ef4ef**

#### Eliminar todos los contenedores detenidos

**$** **docker container rm $(docker container ls -aq)**

1af0bf0ef4ef

108f91112bd4

78ad56b4e93d

1d5649c85450

bbdaf6222610

2ff5d2cd086c

3365d7ec4ec7

#### Mostrar la salida estandar del contenedor en tiempo real (-f)

**$** **docker container logs --details -f 7eefb112ea63**

#### Ejecutar un contenenedor en segundo plano (-d)

**$docker container run --rm -d primera-imagen**

9d17d546b2ef307dbe39d717b70f8e305db982c25edf9481d2b7b6189730008f

#### Información detallada del contenedor

**$docker container inspect hola\_mundo**

#### Unirse a un contenedor en ejecución

**$docker container attach hola\_mundo**

\_apt@7eefb112ea63:/app$

#### Ejecutar un comando en un contenedor en ejecución

**$docker container exec hola\_mundo ls**

main.py

#### Copiar archivos al o desde el contenedor en ejecución

**$docker container cp hola\_mundo:/app/. .**

**$ls**

main.py

**$docker container cp main.py hola\_mundo:/app/main2.py**

**$docker container exec hola\_mundo ls**

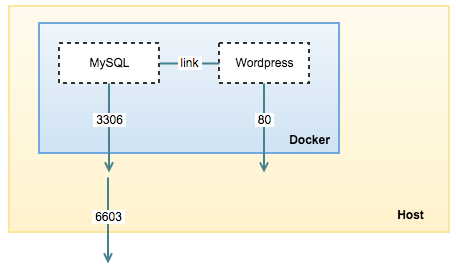
main.py

main2.py

### Gestión de los puertos de comunicación

La comunicación de los contenedores con el resto del sistema suele ser a través de redes de comunicación. Cada contenedor tiene que tener asignada una dirección ip, que se hace de forma automática si no la especificamos y abrir al exterior los puertos que necesite para la comunicación. Los puertos se especificarán por protocolo (tcp/udp) o se utilizará por defecto tcp. Si no se especifican los puertos con el mecanismo que vamos a ver a continuación, solo podrán comunicarse entre sí los contenedores que esté unidos a la misma red, pero no con el exterior.

Para publicar puertos al exterior se usa la opción *-p* de **docker container run**. Con este parámetro hacemos una redirección del puerto del host al puerto del contenedor, con lo que todos los paquetes que lleguen desde el exterior al host al puerto publicado se pasarán de forma automática al puerto del contenedor especificado. Este mecanismo hace que los puertos externos e internos no tengan que ser los mismos, pero sí deberán usar el mismo protocolo. Hay que recordar que solo podremos usar puertos que no estén ya utilizados, no está permitido la reutilización de puertos.



En la imagen vemos dos contenedores, Mysql publica el puerto 3306 internamente que estará conectado al puerto 6603 externo. Wordpress solo publica el puerto interno 80. Por tanto, Wordpres se comunicará con Mysql de forma normal, a través de la red interna y los puertos (80 y 3306). Si algún usuario intentara entrar al puerto 80 del host no pasaría nada, no está conectado con el contenedor, y para acceder desde la red de host a Mysql tendrían que utilizar el puerto 6603.

**$docker container run -p 6603:3306 mysql\_img**

**$docker container run wordpres\_img**

### Políticas de reinicio de contenedores

Un sistema debe ser lo más estable posible, en caso de caída de alguno de los contenedores lo más normal es que sean reiniciados los antes posible para seguir con su labor. Por eso, al ejecutar un contenedor (*run --restart*) se le puede indicar la forma de reinicio del mismo. Por defecto no se reiniciará nunca.

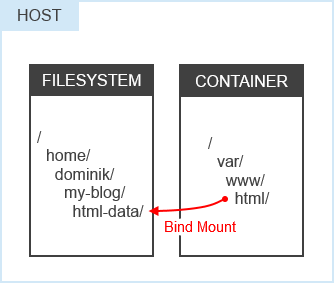
* **no.** Opción por defecto, no se reinicia.
* **on-failure[:num\_reintentos]**. Se reinicia si el contenedor se detiene por un error. Lo reintentará n num\_reintentos antes de parar.
* **always**. Siempre se reinicia a menos que se pare de forma manual. Se reiniciará cuando se reinicie también el servicio Docker.
* **unless-stopped**. Como always pero no se reiniciará si se reinicia el servicio Docker.

## Almacenamiento

Todos los archivos que se crean en un contenedor se almacenan en la última capa que es de lectura – escritura y se desechan cuando el contenedor se elimine, con lo que los perderemos al finalizar. Para evitar la pérdida de datos, Docker implementa dos mecanismos diferentes: Enlaces entre directorios (bind mounts) y volúmenes. Cualquiera de estos mecanismos es persistente y duradero a la eliminación de un contenedor, pero se utilizan en contextos diferentes.

Docker implementa dos métodos más per no los vamos a estudiar en este capítulo.

### Bind mounts

La idea sobre este mecanismo es hacer una unión entre un directorio o fichero del host (externo a Docker) y un directorio o fichero del contenedor, de tal manera que lo que hagamos en cualquiera de los dos puntos se sincronice. Así si copiamos un fichero en un directorio de host enlazado se copia automáticamente dentro del contenedor en el directorio enlazado y viceversa. Cuando se crea este tipo de enlaces, el contenido del directorio del host remplazará al contenido interno.

Para crear este tipo de unión utilizaremos el parámetro *-v* de la orden *run*, indicando el directorio o fichero del host (con la ruta absoluta) y el directorio o fichero del contenedor (con la ruta absoluta). Además, podremos añadir una opción fina *ro* indicando que el enlace es de solo lectura.

**-- En el HOST --**

**$ls**

my\_app

**usuario@ubuntu-docker:~$ ls my\_app/**

main\_bind\_mount.py main.py

**-- En el HOST --**

**$docker container run -it -v $(pwd)/my\_app:/app primera-imagen bash**

**-- En el Contenedor --**

root@d7ad48cb1440:/app# ls

main.py main\_bind\_mount.py

root@d7ad48cb1440:/app# **cp main\_bind\_mount.py main\_bind\_mount\_2.py**

root@d7ad48cb1440:/app# exit

**-- En el HOST --**

**$ls my\_app/**

**main\_bind\_mount\_2.py** main\_bind\_mount.py main.py

El sistema permite crear varios enlaces a través de la misma orden de ejecución simplemente con repetir el comando *v* tantas veces como necesitemos.

### Volúmenes

Un volumen es un almacén de datos gestionado por el sistema Docker, reutilizable entre varios contenedores y persistente a la finalización del contenedor. Cada contenedor puede tener uno o varios volúmenes asociados, dónde guardar los datos que queramos almacenar. A diferencia de los enlaces, no tienen visibilidad externa al contenedor, pero se almacenan en un directorio de host (/var/lib/docker/volumes).

Por las características de los volúmenes, los casos de uso no son los mismos que en el anterior punto. Este tipo de almacenamiento se recomienda para compartir datos entre contenedores, para almacenar los volúmenes en un servidor en la red o para almacenar los datos de los servidores de bases de datos.

**$docker volume**

Usage: docker volume COMMAND

Manage volumes

Commands:

create Create a volume

inspect Display detailed information on one or more volumes

ls List volumes

prune Remove all unused local volumes

rm Remove one or more volumes

Run 'docker volume COMMAND --help' for more information on a command.

Antes de usar un volumen en un contenedor, se recomienda que esté creado e inicializado a través de la orden create, si no se facilita un nombre tendremos que usar su identificador en el resto de órdenes por lo que es recomendable.

**docker volume create *nombre\_volumen***

Una vez creado debemos unirlo a un directorio del contrenedor, se hace en el momento de la ejecución (*run*) mediante el parámetro *-v* la igual que los bind mounts, pero la diferencia es que el primer parámetro ahora será un nombre de volumen o identificador en vez de una ruta a un directorio o archivo.

**$docker volume create mi\_volumen**

mi\_volumen

**$docker volume ls**

DRIVER VOLUME NAME

local mi\_volumen

**$docker container run -it -v mi\_volumen:/app primera-imagen bash**

root@07f5ca4fb4c0:/app# ls

main.py

root@07f5ca4fb4c0:/app# cp main.py main\_volumen.py

root@07f5ca4fb4c0:/app# ls

main.py main\_volumen.py

root@07f5ca4fb4c0:/app# exit

**$docker container run -it -v mi\_volumen:/app primera-imagen bash**

root@07f5ca4fb4c0:/app# ls

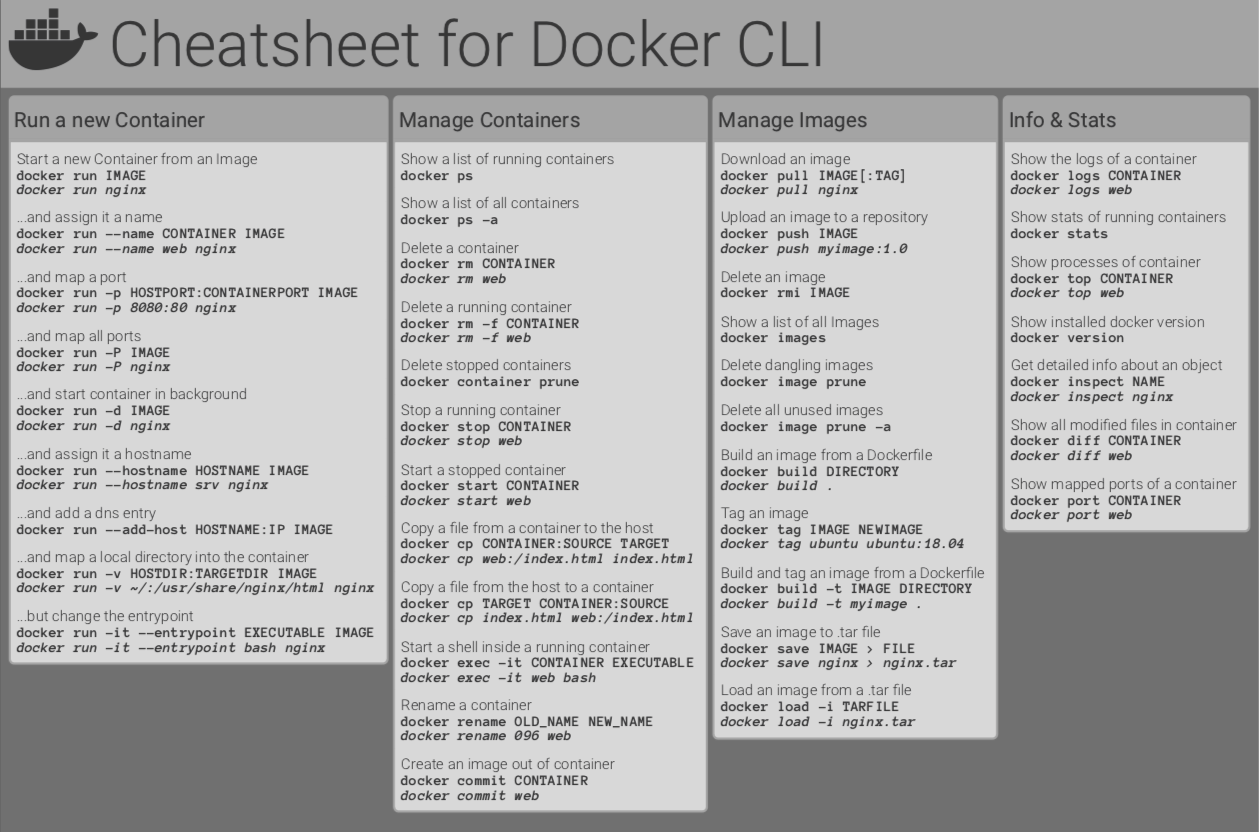
main.py main\_volumen.py

La gestión de volúmenes es más extensa, incluyendo diferentes drivers para los volúmenes que permiten su almacenamiento en sistemas remotos (*https://docs.docker.com/storage/*)

### Órdenes de gestión

|  |  |  |
| --- | --- | --- |
| Orden | Descripción | Parámetros |
| docker volume rm | Borra un volumen del sistema, n o debe de esta en uso | -f forzar la eliminación |
| docker volume ls | Listado de los volúmenes existentes en el host local |  |
| docker volume prune | Borra todos los volúmenes que no están en uso | -f no pide confirmación |
| docker volume inspect | Información de un volumen en concreto |  |

## Resúmenes de comandos sintaxis antigua



### Comandos interesantes

#### Ejecutar un shell como usuario root

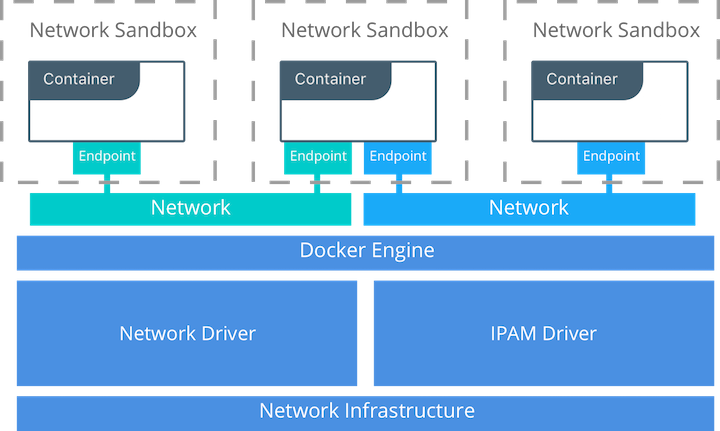
**$docker run -it --name test --user 0 primera-imagen bash**

#### Borrar todas las imágenes existentes, solo bajo Linux

**docker rmi -f $(docker images -aq)**

## Redes bajo Docker

### Arquitectura



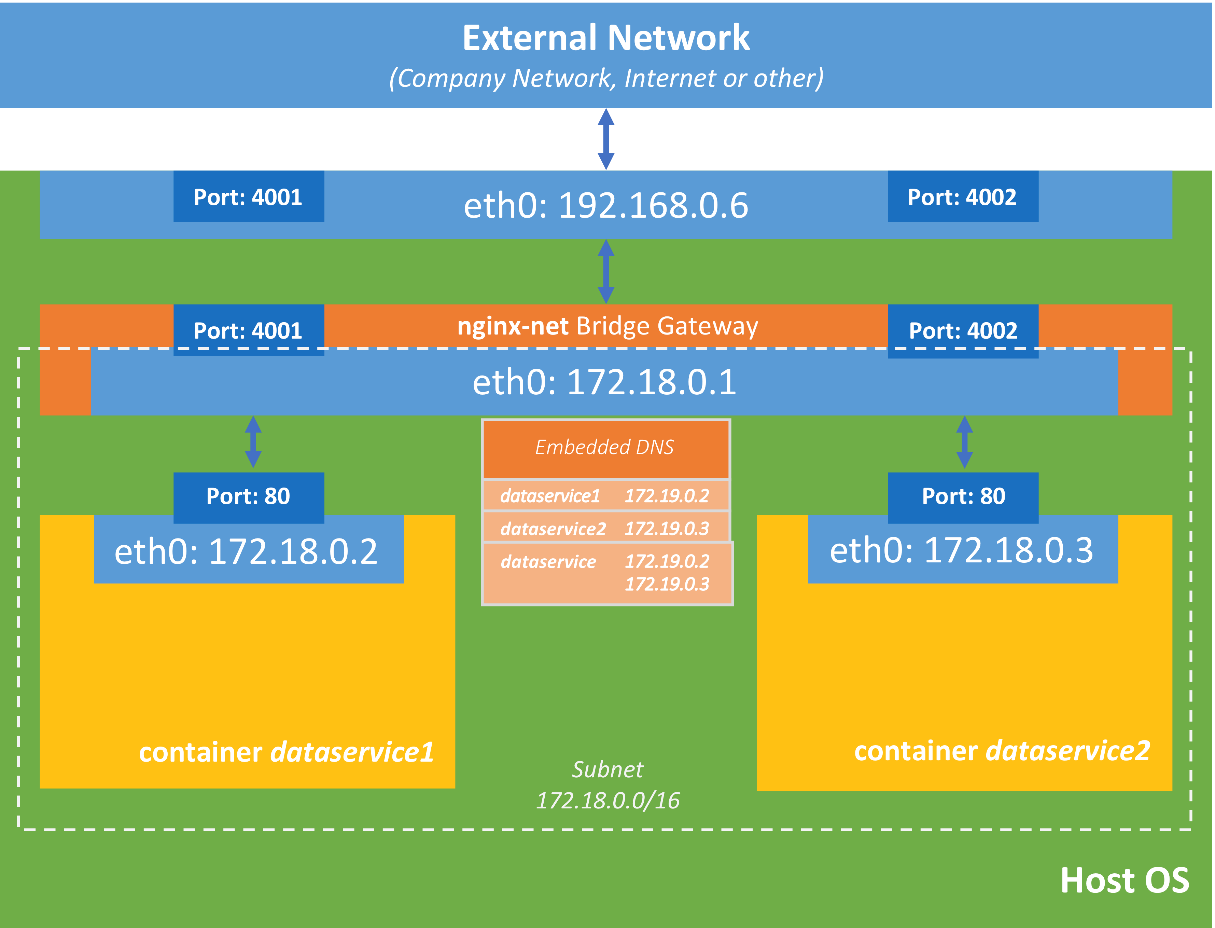
La arquitectura de red implementada dentro de Docker la encontramos en la imagen anterior. De ella podemos destacar tres elementos: Sandbox, Endpoint y Network. Estos son los componentes básicos que vamos a describir a continuación.

* **Sandbox**. Es la implementación de la red dentro de un contenedor, está formado por el software de comunicaciones del contenedor y la configuración necesaria. Cada contenedor tiene una Sandbox diferente.
* **Endpoint**. Es una interfaz de red para el contenedor, una tarjeta de red. Puede tener tantas como necesitemos para el contenedor.
* **Network**. Es un conjunto de Endopoints que se pueden comunicar entre sí a través de la red. Es una red lógica dentro del sistema.

### Drivers de red

El tipo de redes que se pueden crear depende del driver que utilicemos en el momento de la creación de la red. Cada driver permite unas facilidades u otras. Vamos a ver los más comunes.

* **bridge.** Es el driver por defecto que se usará al crear si no se especifica. Permite comunicarse a contenedores internos entre sí conectados a la misma red y conectarse con el host mediante mapeo de puertos. El sistema Docker crea una red por defecto de este tipo que será la utilizada en el momento de la ejecución de un contenedor si no especificamos ninguna.



Este tipo de redes se comportan como redes privadas que permiten que los contenedores que se están ejecutando en el mismo host de Docker pueden comunicarse entre sí. Los contenedores que pertenecen a la misma red bridge tienen todos los puertos accesibles entre ellos, sin embargo, es obligatorio publicarlos (*-p*) para que estén accesibles desde el exterior. El sistema crea una red por defecto a la que se conectan todos los contenedores si no se especifica una red de usuario, pero se recomienda crear redes de usuario para entornos de producción.

* **host**. Este tipo de red permite compartir espacio de direcciones con el host, con lo que el contenedor será accesible a través de la ip del host y los puertos de este. Solo puede existir una red de este tipo.
* **overlay**. Permite crear una red distribuida entre host Docker.
* **null**. Indicará que no hay red.

### Gestión de las redes: docker network

**$docker network**

Usage: docker network COMMAND

Manage networks

Commands:

connect Connect a container to a network

create Create a network

disconnect Disconnect a container from a network

inspect Display detailed information on one or more networks

ls List networks

prune Remove all unused networks

rm Remove one or more networks

Run 'docker network COMMAND --help' for more information on a command.

|  |  |  |
| --- | --- | --- |
| Orden | Descripción |  |
| docker network connect | Conectar un contenedor a una red | |
| docker network create | Crea una red privada | |
| docker network disconect | Desconecta un contenedor de una red | |
| docker network inspect | Muestra información detallada de la red | |
| docker network ls | Lista las redes existentes |  |
| docker network prune | Elimina todas las redes no utilizadas | |
| docker network rm | Borra una red |  |

### Creación de redes: docker network create

**$docker network create --help**

Usage: docker network create [OPTIONS] NETWORK

Create a network

Options:

--attachable Enable manual container attachment

--aux-address map Auxiliary IPv4 or IPv6 addresses used by Network driver (default map[])

--config-from string The network from which to copy the configuration

--config-only Create a configuration only network

-d, --driver string Driver to manage the Network (default "bridge")

--gateway strings IPv4 or IPv6 Gateway for the master subnet

--ingress Create swarm routing-mesh network

--internal Restrict external access to the network

--ip-range strings Allocate container ip from a sub-range

--ipam-driver string IP Address Management Driver (default "default")

--ipam-opt map Set IPAM driver specific options (default map[])

--ipv6 Enable IPv6 networking

--label list Set metadata on a network

-o, --opt map Set driver specific options (default map[])

--scope string Control the network's scope

--subnet strings Subnet in CIDR format that represents a network segment

|  |  |  |
| --- | --- | --- |
| Orden | Parámetro | Descripción |
| docker network create | --attachable | Permite que se puedan conectar contenedores de forma manual a una red overlay definida por el usuario en Docker Swarm |
|  | --config-only | Crea una red para ser usada para configuraciones de otras redes, no se pueden añadir contenedores. |
|  | -d | Driver de la red que estamos creando |
|  | --gateway strings | Puerta de enlace de la red |
|  | --ingress | Gestión de la red dentro de nodos clusters |
|  | --internal | No puede tener comunicaciones externas |
|  | --ip-range strings | Rango de ips que se usarán para los contenedores |
|  | --ipv6 | Habilita la red con ip6 |
|  | -o | Opciones de los drivers de red |
|  | --scope string | Indica el ámbito de la red: swarm, global o local |
|  | --subnet string | Define una subred en formato CDIR, las redes bridge solo podrán tener una subred |

**$docker network create -d bridge red\_interna**

### Conectar un contenedor a una red: docker network connect

Para conectar un contenedor a una red se puede usar este comando o la opción *--network* de la orden *run*.

**$docker network connect red\_interna primera-imagen**

Las opciones para el parámetro *run* son:

* --dns. Para configurar el DNS
* --network nombre. Nombre de la red.
* --ip dir. Dirección ip del contenedor.
* --ip6 dir. Dirección ip6 del contenedor.

**$docker run -it --name test --network red\_interna –ip 192.168.1.2 primera-imagen**

### Ejemplo de uso: Mysql – WordPress – PhpMyAdmin

**$docker network create wordpress\_net**

cec2190a46f0157e0eb3ade47e580c734e508c24a9c69b40b5eea757ddb54180

**$docker volume create mysql\_data**

mysql\_data

**$docker run -d \**

**--rm \**

**--name mysql\_c \**

**--network wordpress\_net \**

**-e MYSQL\_ROOT\_PASSWORD=root \**

**-e MYSQL\_DATABASE=wp\_database \**

**-e MYSQL\_USER=wp\_user \**

**-e MYSQL\_PASSWORD=wp\_password \**

**-v mysql\_data:/var/lib/data \**

**mysql:8.0**

10f5e7b811b6c56f9f354578ef45b2ec5d2e6bba2384b6da9e4d9ae59e57b436

**$docker volume create wordpress\_data**

wordpress\_data

**$docker run -d \**

**--rm \**

**--network wordpress\_net \**

**-p 80:80 \**

**-e WORDPRESS\_DB\_HOST=mysql\_c \**

**-e WORDPRESS\_DB\_NAME=wp\_database \**

**-e WORDPRESS\_DB\_USER=wp\_user \**

**-e WORDPRESS\_DB\_PASSWORD=wp\_password \**

**-v wordpress\_data:/var/www/html \**

**wordpress:php8.0**

a7948ad7620386c6ba29e57a503de47e426f026e331fe0b9e6953f8c06f5a27a

**$docker run -d \**

**--rm \**

**--network wordpress\_net \**

**-p 8080:80 \**

**-e PMA\_HOST=mysql\_c \**

**phpmyadmin:5.1.1**

953581487e9fe2b61cad3a429d1fbe8df54615868bf9f7eb269d6e7667bff900

Hay que tener cuidado con los espacios en blanco entre los iguales, no debe quedar ninguno.

**$docker container ls**

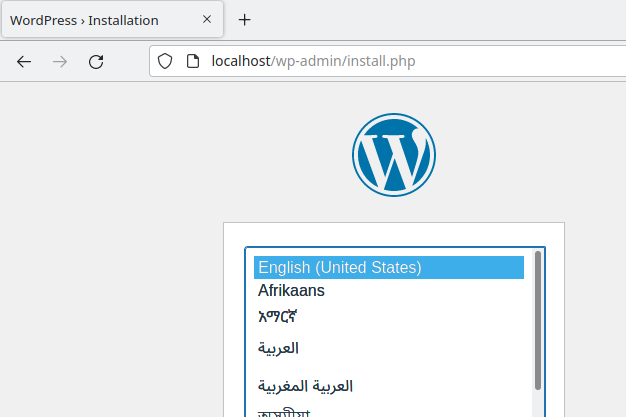
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

a7948ad76203 wordpress:php8.0 "docker-entrypoint.s…" 40 seconds ago Up 39 seconds 0.0.0.0:80->80/tcp, :::80->80/tcp funny\_northcutt

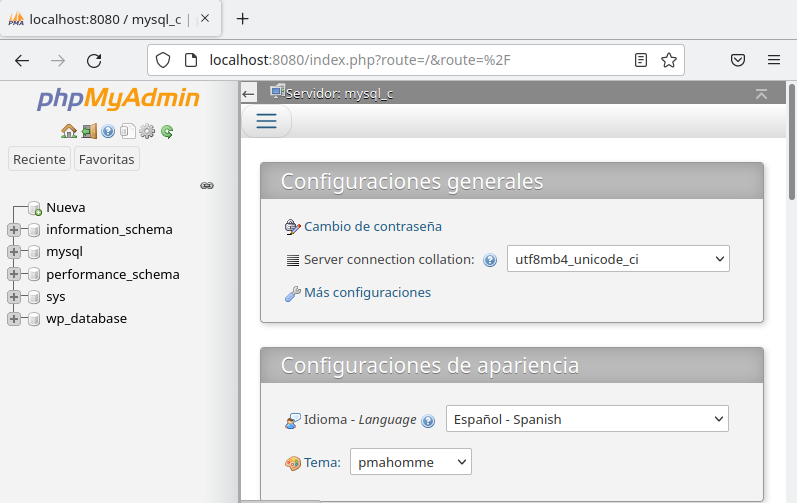
953581487e9f phpmyadmin:5.1.1 "/docker-entrypoint.…" About a minute ago Up About a minute 0.0.0.0:8080->80/tcp, :::8080->80/tcp nifty\_cori

10f5e7b811b6 mysql:8.0 "docker-entrypoint.s…" 9 minutes ago Up 9 minutes 3306/tcp, 33060/tcp mysql\_c

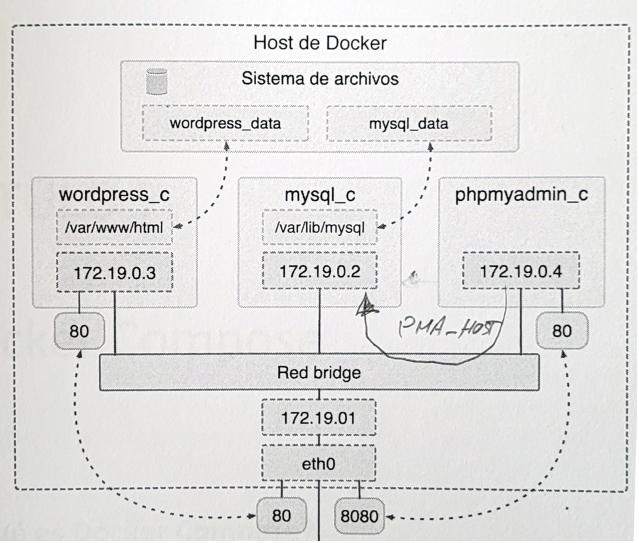
#### Acceso a wordpress



#### Acceso a phpmyadmin



#### Estructura de la aplicación



## Docker compose

La gestión individual de contenedores se puede llevar a cabo cuando hay uno o muy pocos, cuando el número crece y las dependencias entre ellos son relevante, tenemos que hacer uso de esta herramienta. Docker compose permite gestionar y ejecutar aplicaciones con varios contenedores. Una vez definida la aplicación, podremos crearlos e iniciarlos con un único comando. Docker implementa la gestión a través de dos comandos diferentes, uno implementado bajo Python: ***docker-compose***, instalado por defecto y otro **d*ocker compose*** implementado bajo Go que hay que instalar.

### El archivo de configuración

Al igual que Docker, Docker compose utiliza uno o varios ficheros de configuración para su ejecución, utilizando el formato YAML (no se pueden usar tabuladores, hay que usar espacios en blanco). El nombre de este archivo por defecto es docker-compose.yml o docker-compose.yaml.

El archivo está formado por seis secciones, pero la única obligatoria es la que describe los servicios a ejecutar y su orden.

* **version**. Indica la versión del archivo que estamos utilizando, su función es meramente informativa. Es una cadena de texto, por lo que debe ir encerrada entre comillas: ‘3.8’. La versión que usemos indicará las opciones disponibles en el fichero.
* **services.** Define los servicios y sus relaciones, contendrá al menos un servicio y aparecerá un servicio por cada contenedor con sus opciones. Las opciones que pueden tener incluyen:

|  |  |
| --- | --- |
| Parámetro | Descripción |
| image | Definir la imagen que se usará para crear el contenedor |
| build | Ruta al fichero Dockerfile del contenedor |
| command | Permite sobrescribir el CMD del Dockerfile |
| entrypoint | Permite sobrescribir el ENTRYPOINT del Dockerfile |
| container\_name | Nombre del contenedor |
| ports | Puertos con la sintaxis -p del comando run |
| volumes | Volúmenes, similar a -v del comando run |
| networks | Redes de usuario, similar a –network del comando run |
| enviroment | Lista de variables de entorno y sus valores |
| env\_file | Ruta al directorio de variables de entorno |
| depends\_on | Nombre de los servicios que deben estar antes en ejecución para lanzarlo |
| restart | Política de reinicio del contenedor |

* **volumes.** Define los espacios de almacenamiento de los servicios. Los volúmenes que aquí se especifique se crearán de forma automática. Es obligatorio definir en esta sección los volúmenes si luego los vamos a usar en la sección services, pero no así los bind mount.
* **networks**. Define las redes de usuario de la aplicación que posteriormente se usaran en la sección services con la opción network.

### Gestión de docker compose: docker-compose

**$sudo apt-get install docker-compose**

**$docker-compose**

Define and run multi-container applications with Docker.

Usage:

docker-compose [-f <arg>...] [options] [--] [COMMAND] [ARGS...]

docker-compose -h|--help

Options:

-f, --file FILE Specify an alternate compose file

(default: docker-compose.yml)

-p, --project-name NAME Specify an alternate project name

(default: directory name)

-c, --context NAME Specify a context name

--verbose Show more output

--log-level LEVEL Set log level (DEBUG, INFO, WARNING, ERROR, CRITICAL)

--no-ansi Do not print ANSI control characters

-v, --version Print version and exit

-H, --host HOST Daemon socket to connect to

--tls Use TLS; implied by --tlsverify

--tlscacert CA\_PATH Trust certs signed only by this CA

--tlscert CLIENT\_CERT\_PATH Path to TLS certificate file

--tlskey TLS\_KEY\_PATH Path to TLS key file

--tlsverify Use TLS and verify the remote

--skip-hostname-check Don't check the daemon's hostname against the

name specified in the client certificate

--project-directory PATH Specify an alternate working directory

(default: the path of the Compose file)

--compatibility If set, Compose will attempt to convert keys

in v3 files to their non-Swarm equivalent (DEPRECATED)

--env-file PATH Specify an alternate environment file

Commands:

build Build or rebuild services

config Validate and view the Compose file

create Create services

down Stop and remove containers, networks, images, and volumes

events Receive real time events from containers

exec Execute a command in a running container

help Get help on a command

images List images

kill Kill containers

logs View output from containers

pause Pause services

port Print the public port for a port binding

ps List containers

pull Pull service images

push Push service images

restart Restart services

rm Remove stopped containers

run Run a one-off command

scale Set number of containers for a service

start Start services

stop Stop services

top Display the running processes

unpause Unpause services

up Create and start containers

version Show version information and quit

|  |  |  |
| --- | --- | --- |
| Orden | Descripción | Parámetros |
| docker-compose up | Iniciar todos los contenedores | -d ejecutarlos en segundo plano |
| docker-compose down | Detiene todos los contenedores |  |
| docker-compose ps | Estado de la aplicación |  |
| docker-compose logs | Permite visualizar los logs de los contenedores | -f hace que se muestren en tiempo real |
| docker-compose exec | Ejecuta un comando en un contenedor |  |
| docker-compose build | Crea las imágenes de las que dependen los contenedores |  |

### Ejemplo de aplicación: Mysql – WordPress – PhpMyAdmin

**mysql.env**

MYSQL\_ROOT\_PASSWORD=root

MYSQL\_DATABASE=wp\_database

MYSQL\_USER=wp\_user

MYSQL\_PASSWORD=wp\_password

**docker-compose.yml**

version: "3.8"

services:

wordpress:

image: wordpress:php8.0

ports:

- "80:80"

environment:

- WORDPRESS\_DB\_HOST=mysql

- WORDPRESS\_DB\_NAME=${MYSQL\_DATABASE}

- WORDPRESS\_DB\_USER=${MYSQL\_USER}

- WORDPRESS\_DB\_PASSWORD=${MYSQL\_PASSWORD}

volumes:

- wordpress\_data:/var/www/html

depends\_on:

- mysql

restart: always

networks:

- wordpress\_net

mysql:

image: mysql:8.0

environment:

- MYSQL\_ROOT\_PASSWORD=${MYSQL\_ROOT\_PASSWORD}

- MYSQL\_DATABASE=${MYSQL\_DATABASE}

- MYSQL\_USER=${MYSQL\_USER}

- MYSQL\_PASSWORD=${MYSQL\_PASSWORD}

volumes:

- mysql\_data:/var/lib/data

restart: always

networks:

- wordpress\_net

phpmyadmin:

image: phpmyadmin:5.1.1

ports:

- "8080:80"

environment:

- PMA\_HOST=mysql

depends\_on:

- mysql

restart: always

networks:

- wordpress\_net

volumes:

mysql\_data:

wordpress\_data:

networks:

wordpress\_net:

**$docker-compose --env-file mysql.env up**

^C

**$docker-compose down**

# Capítulo V. Análisis y diseño de los ms

## Análisis del sistema monolítico

### Análisis de páginas y datos

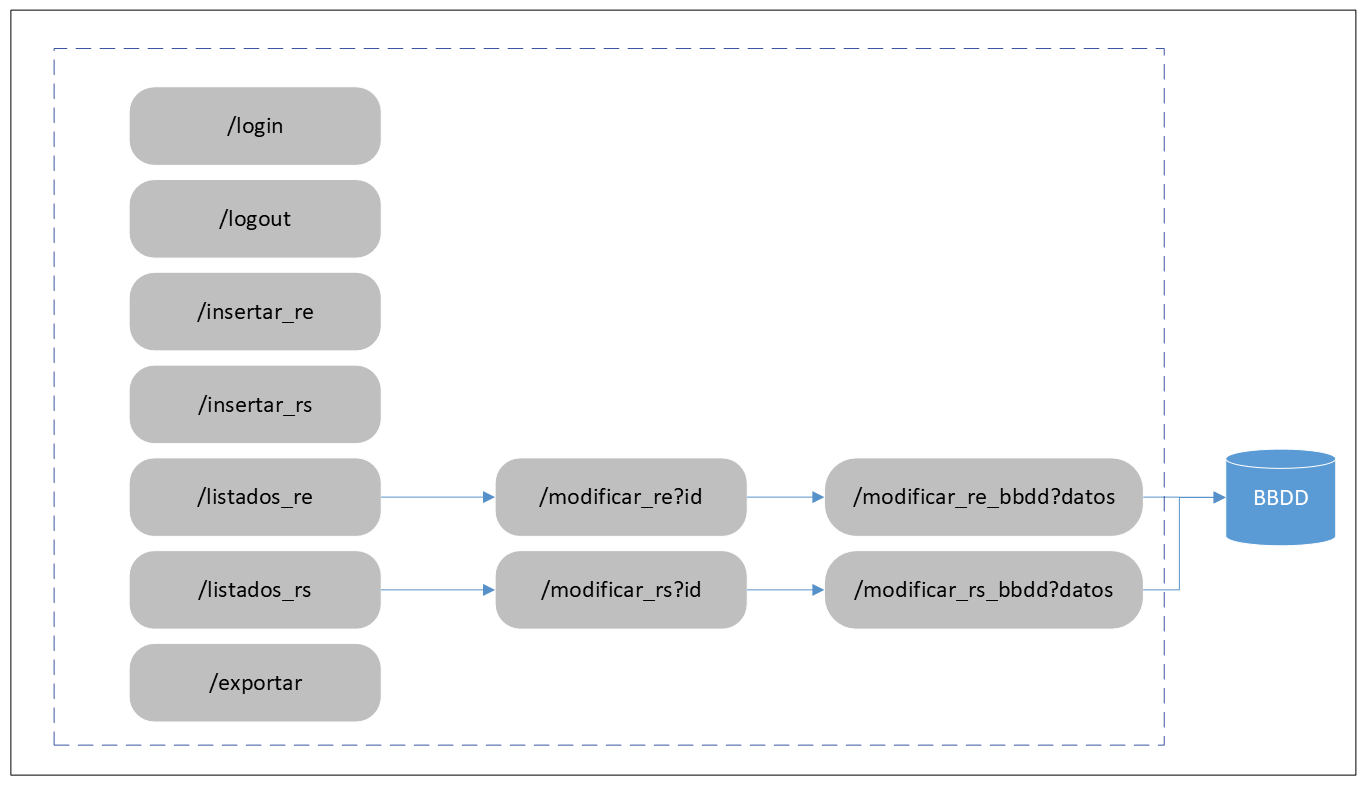


Ilustración Movimiento de ventanas

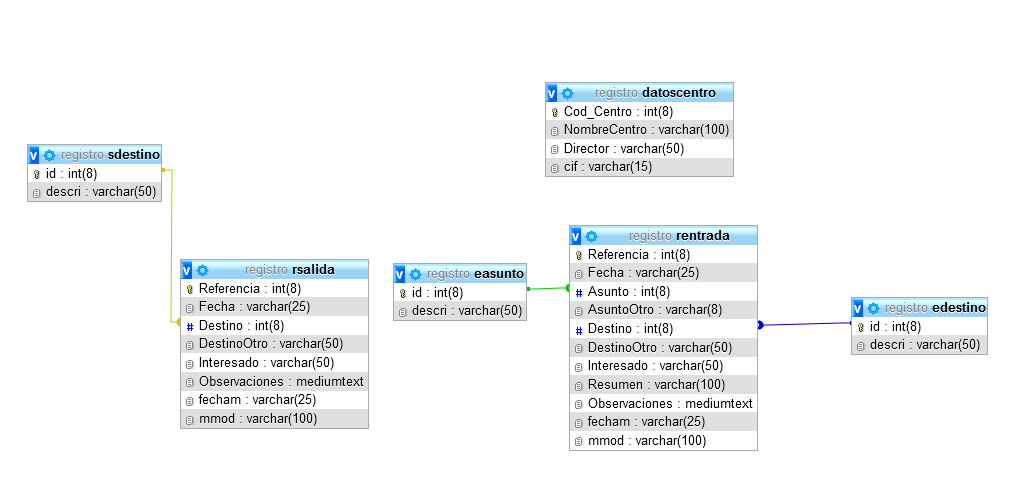


Ilustración Estructura de la BBDD

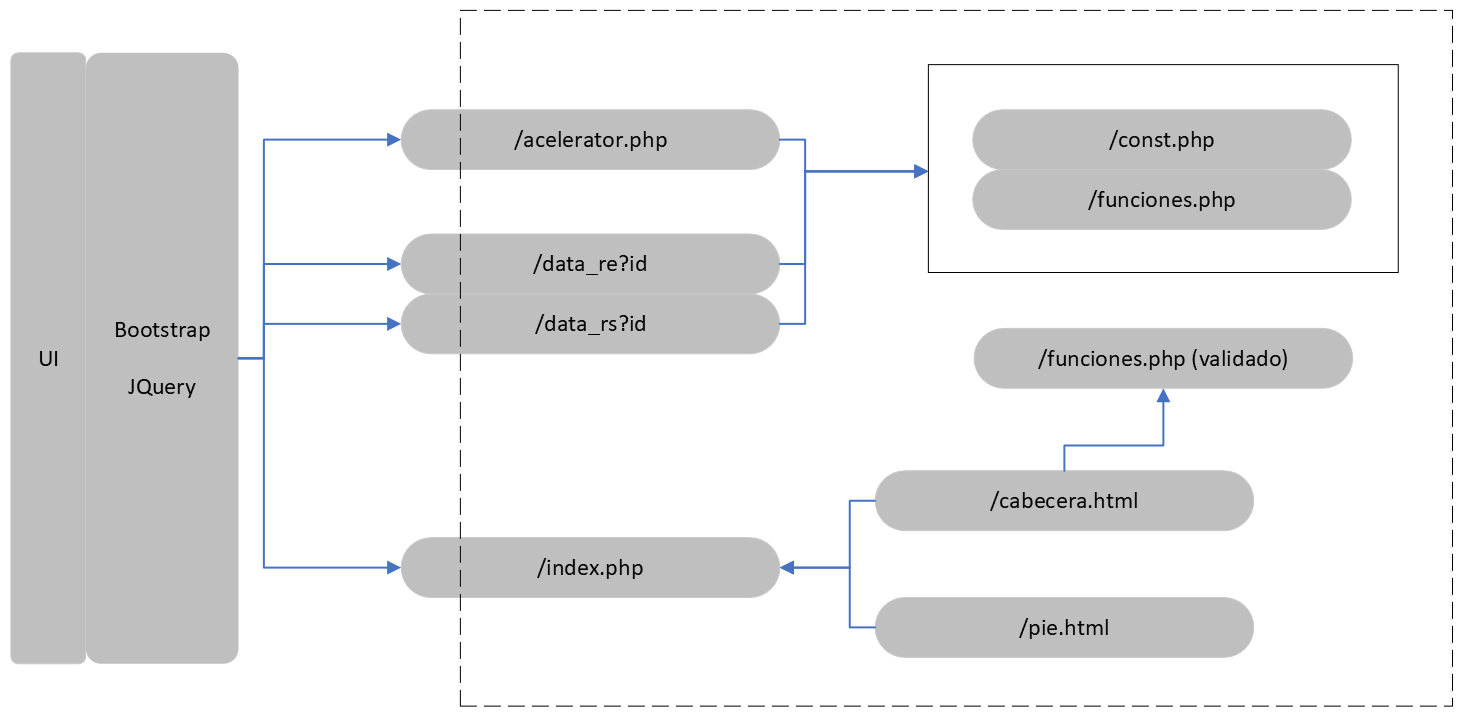
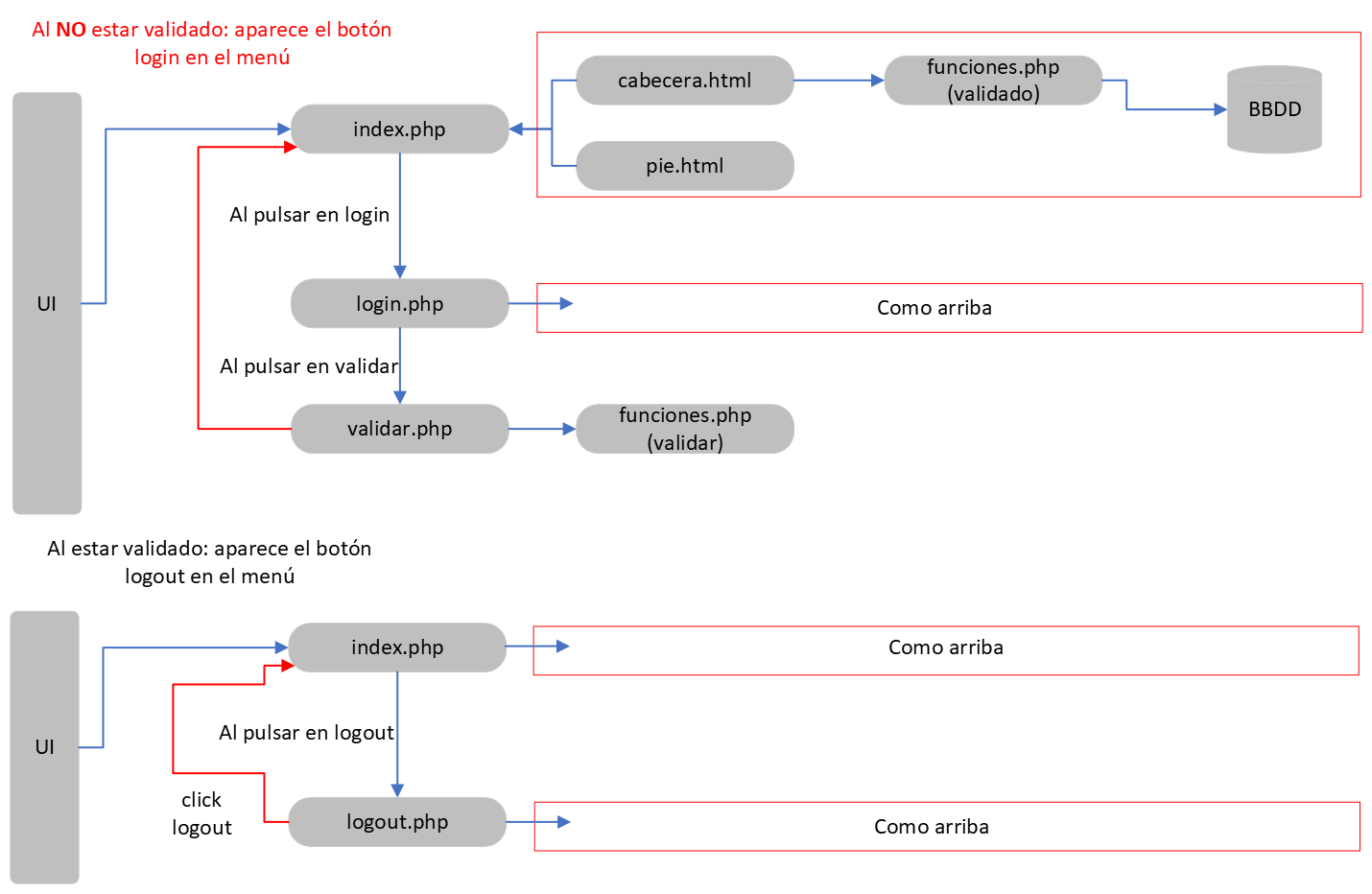


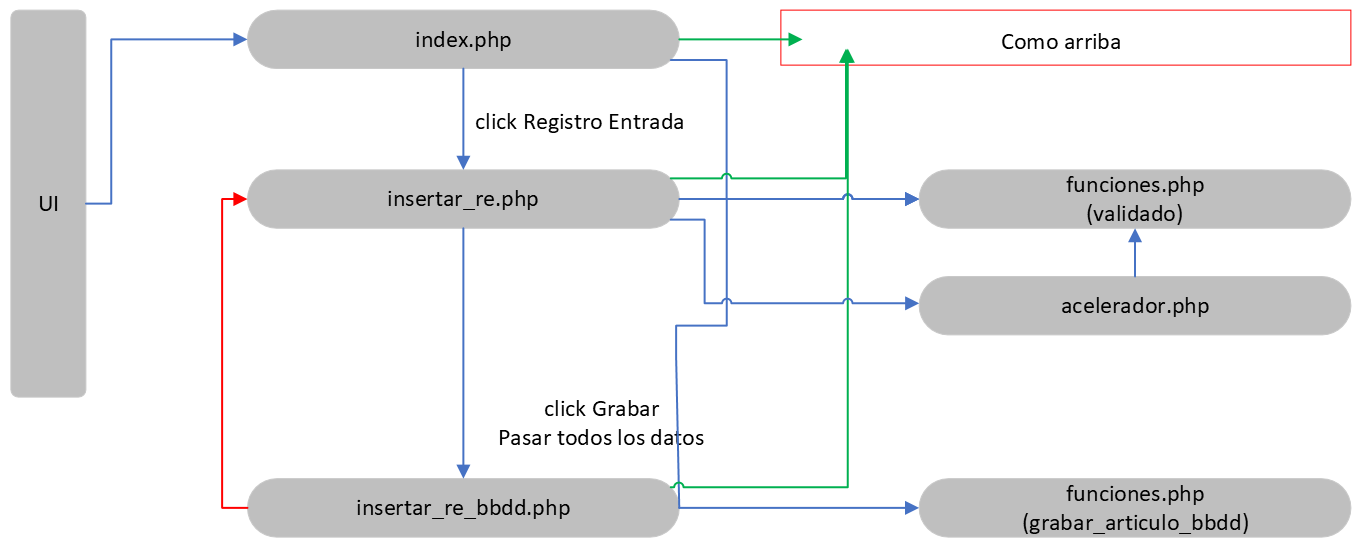
Ilustración Visualización de datos

### Análisis del funcionamiento: login - logout



En esta transición se envía por parte del usuario la clave que será recogida por el fichero **validar.php** y validará o no el usuario en la función **validar()** dejando registro en la sesión para poder controlar la validación.

### Análisis del funcionamiento: Registro de entrada

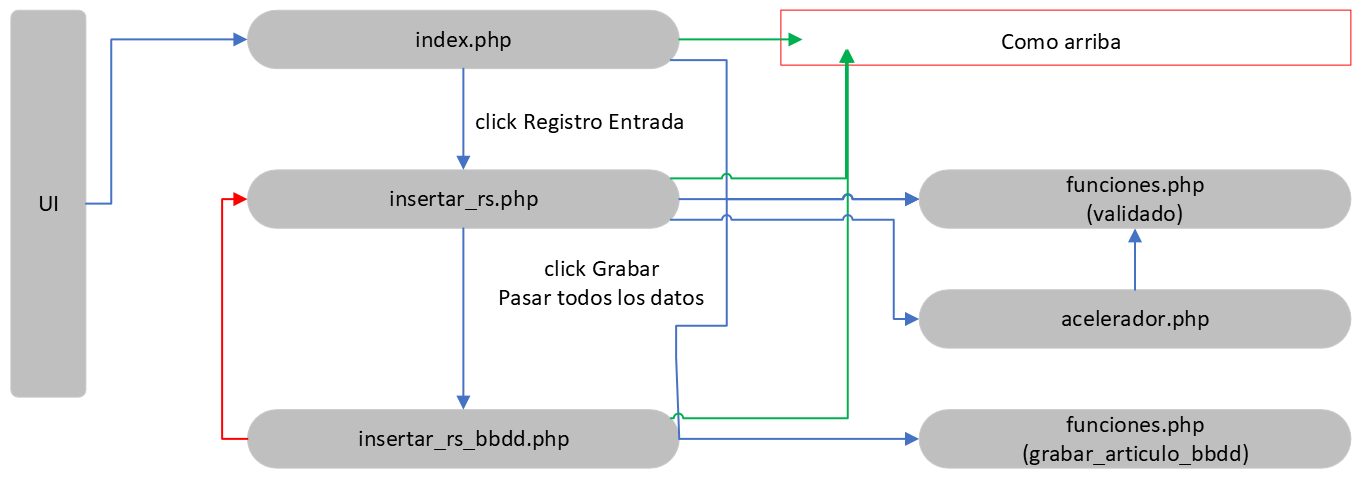


Se carga un formulario vacío con todos los datos necesarios del registro de entrada, solo en caso que se esté validado (funciones.php/validado()). Este formulario mostrará una lista en su campo interesado según se va escribiendo, el contenido se usará como filtro para mostrar aquellos nombres que comiencen por dicho texto. Cuando estén rellenos los campos obligatorios, se podrá grabar.

Los campos Asunto y Destino se cargan en una combo con los valores de las tablas correspondientes (funciones.php/get\_table\_data\_select()).

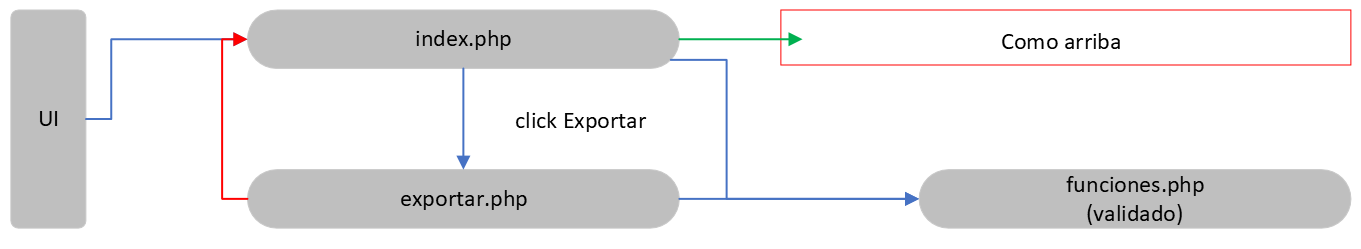
En caso de grabar se pasará el formulario a la página insertar\_re\_bbdd que en caso de estar validado almacenará los datos en la BBDD mediante funciones.php/grabar\_Articulo\_bbdd(). Esta función reflejará en la sesión que se ha grabado de forma correcta o no para en la vuelta a la página insertar\_re.php se muestre en la cabecera el mensaje de feed back.

### Análisis del funcionamiento: Registro de salida



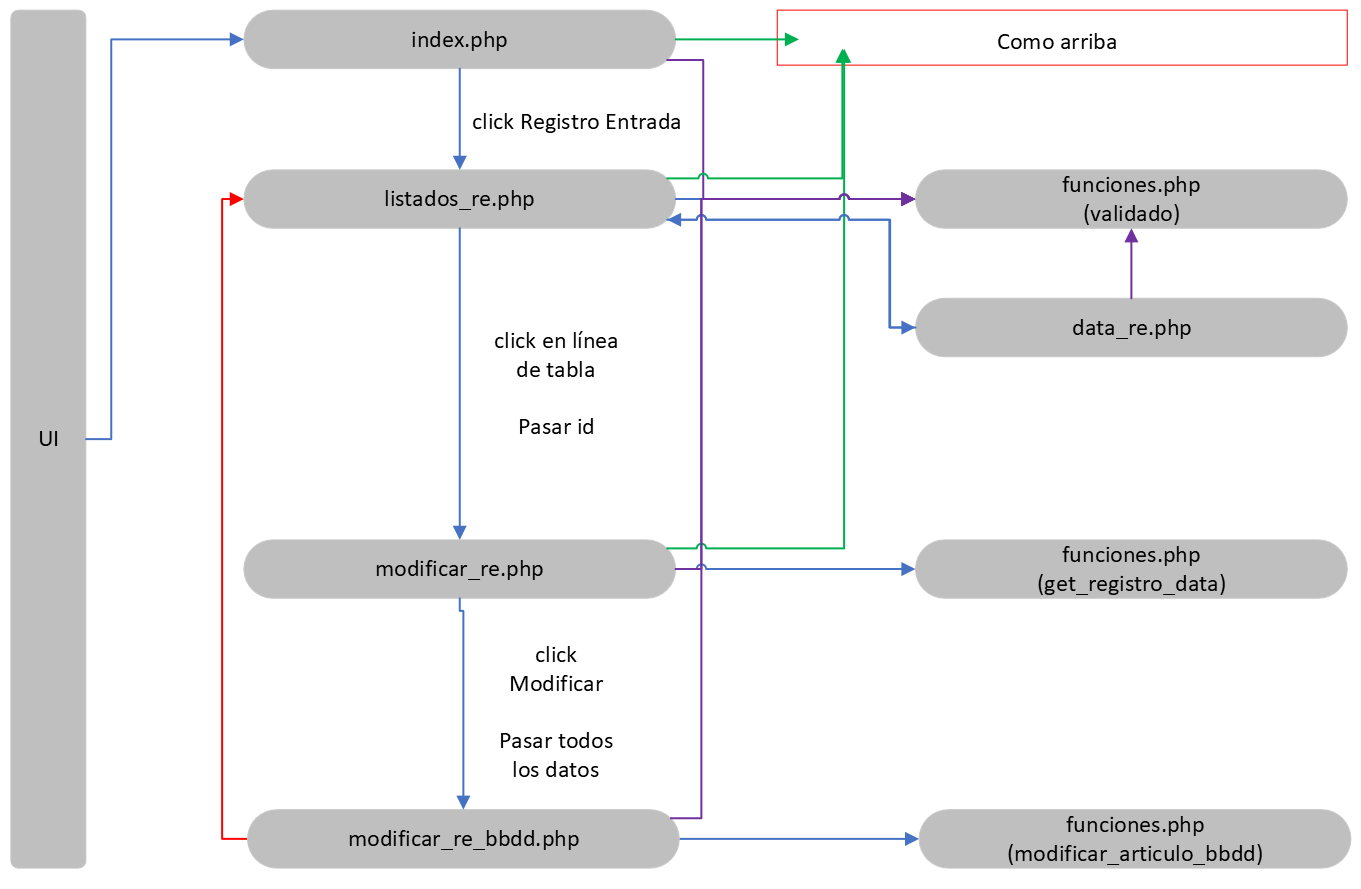
El funcionamiento de esta parte de la aplicación es similar a la del registro de entrada, solo cambia el formulario que se presenta, al ser necesarios menos campos que en el registro de entrada. El resto es similar.

### Análisis del funcionamiento: Exportar



En caso de estar validado el usuario se descarga un fichero Excel con todo el contenido de las BBDD.

### Análisis del funcionamiento: Listados registro entrada



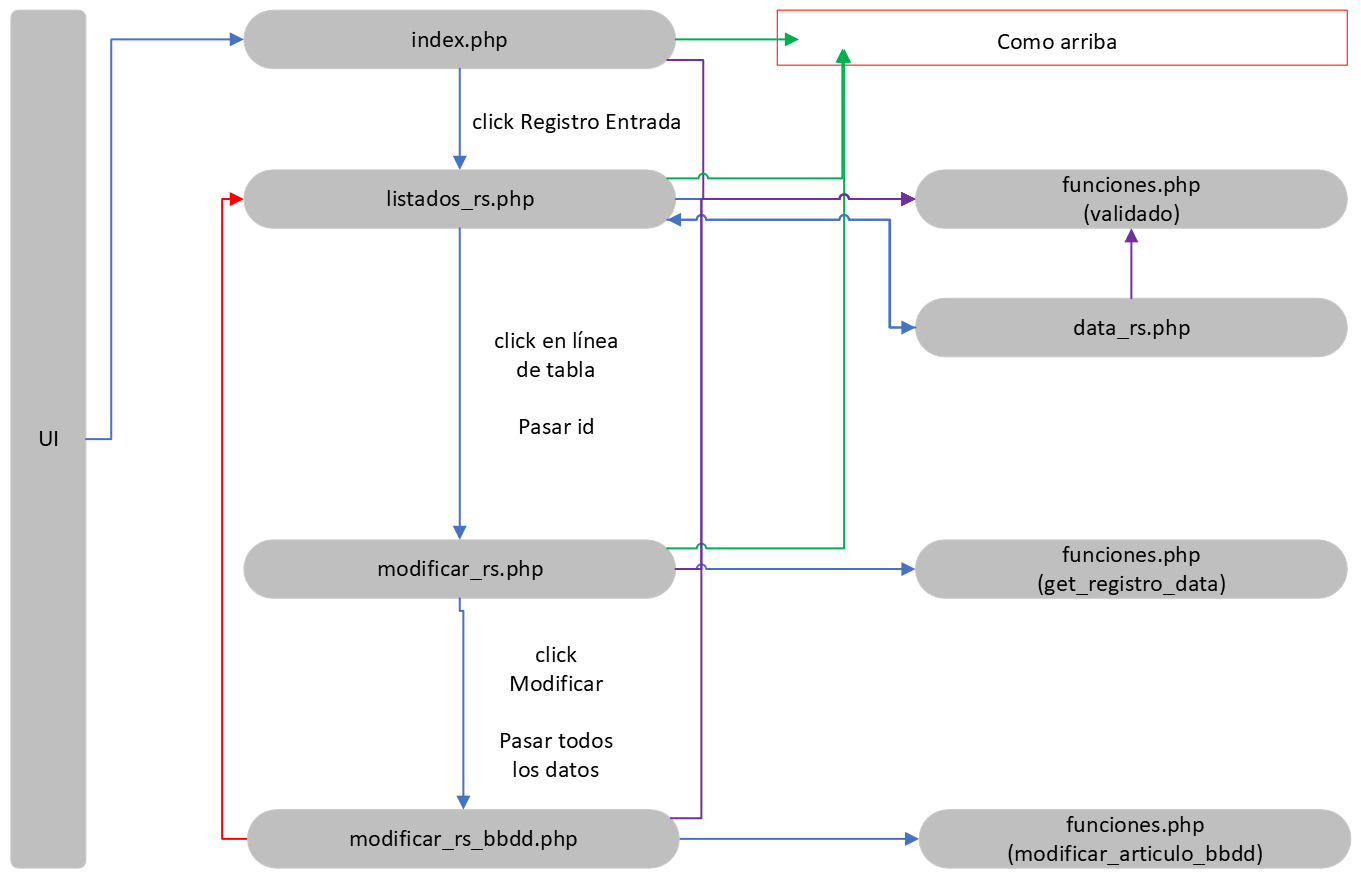
Si el usuario está validado, se cargará la lista central con todos los datos del registro de entrada obtenidos desde la página data\_re.php. La tabla que se presenta se puede ordenar por columnas, filtrar, exportar a pdf y mandar a la impresora.

Si pulsamos en cualquier fila, se llama a la página modificar\_re.php que mostrará los datos del registro en el que se haya pulsado (estos datos se obtienen con funciones.php/get\_registro\_data()) pudiéndose modificar todos los valores excepto la referencia.

Los campos Asunto y Destino se cargan en una combo con los valores de las tablas correspondientes (funciones.php/get\_table\_data\_select()).

En caso que se pulse a modificar se mandarán todos los datos a la página modificar\_re\_bbdd que grabará en la BBDD las modificaciones a través de funciones.php/modificar\_articulo\_bbdd().

### Análisis del funcionamiento: Listados registro salida



La única diferencia con el punto anterior es que los datos que se presentan son los del registro de salida.

## Diseño del sistema basado en ms



Ilustración Diseño de nuestro sistema en ms

Tras el estudio detallado de la arquitectura monolítica existente, hemos decidido crear la estructura anterior de ms para dar servicio a la nueva aplicación. Hemos comprobado que los límites existentes entre estos ms son muy fuertes, no comparte ni funcionalidad ni datos entre ninguno de ellos, siendo relativamente “sencilla” esta división.

* **Autorización**. Este ms será el encargado de validar los usuarios y de distribuir los tokens necesarios para realizar las operaciones en el sistema. En el funcionamiento básico será el siguiente. Un usuario con contraseña y clave se validará en el servicio. Este devolverá un token al usuario en caso de validación correcta que tendrá una vida máxima de un día.

Este token será enviado por el usuario en cada petición al resto de ms. Cuando un ms recibe una petición de servicio, se conecta con el servicio de autorización para validar el token. En caso que sea validado, se pasa a realizar la acción, en otro caso devolverá un error.

* **Rentrada**. Este ms se encargará de gestionar el registro de entrada. Recibirá las peticiones por parte del usuario, validará la petición y devolverá el resultado.
* **Rsalida**. Similar al anterior, pero con el registro de salida.

En el estudio detallado de la aplicación monolítica descubrimos que no existía un interfaz de administración y que el interfaz de usuario no tenía una estructura en capas, con lo que se decide **realizar un remplazo completo de la aplicación, no convivirán en ningún momento ambas versiones en producción**.

### Desarrollo temporal

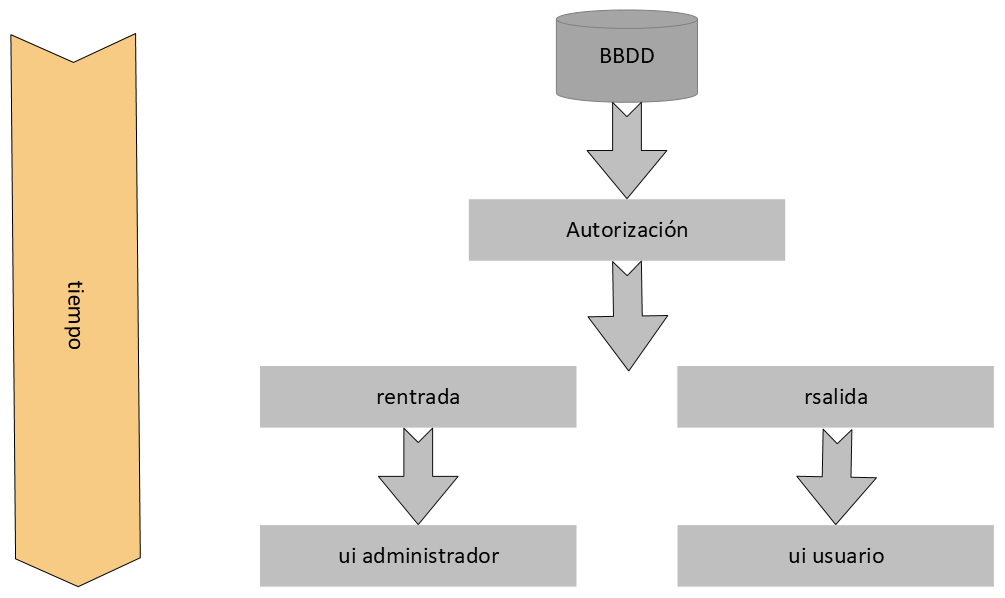
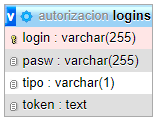


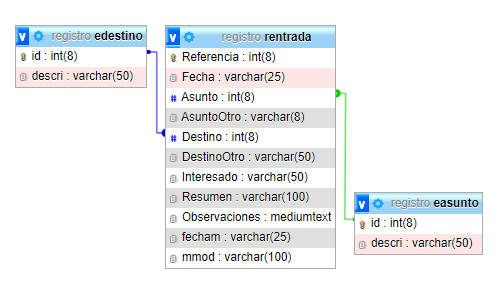
Ilustración Desarrollo Temporal

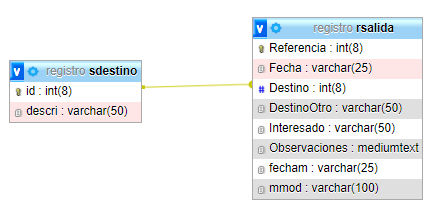
En un entorno ideal, y para un aula que está pensado este problema, se asignará a una persona el ms de autorización, a otra persona el registro de entrada y el interfaz de administrador y a otra el registro de salida y el UI de usuario. Además, la persona encargada del ms de autorización será la encargada de realizar los test de aceptación de toda la aplicación, trabajando sobre un entorno tipo GitHub para la gestión del software.

El diseño de la BBDD está en el primer capítulo de este libro, solo hay que implementarlo en tres BBDD diferentes para un escalado correcto: **Auth**, **Rentrada**, **Rsalida**.



Al sustituir una aplicación completamente podemos permitirnos ciertas licencias, en este caso vamos a añadir a nuestra aplicación la capacidad de gestión por parte de un administrador de esta tabla, con lo que aparece un nuevo campo (**tipo**) que determinará si es un usuario normal (U) o un usuario administrador (A).





### Arquitecturas elegidas

Para desarrollar nuestra aplicación utilizaremos tecnologías basadas en Python, en concreto para los servicios de autorización, rentrada y rsalida usaremos Flask junto con el framework RestX un fork de restplus (<https://flask-restx.readthedocs.io/en/latest/>, pip install flask-restx), para el acceso a las bases de datos usaremos el framework SQLAlchelmy. Los interfaces de usuario se desarrollarán bajo HTML, CSS y javascript dejando para el alumno mejorar este interfaz con JQUERY y Bootstrap.

Cada ms (excepto la bbdd) será desplegado como un contenedor Docker en el que se instalará un servidor NGINX. En una primera aproximación no será necesario el uso de Kubernetes para un despliegue mayor, en una segunda fase se verá esta aproximación.

## Diseño ms autorización

|  |  |
| --- | --- |
| **Nombre** | autorizacion |
| **Dirección** | localhost |
| **Puerto** | 15000 |

En todo entorno distribuido hace falta una gestión de autorizaciones, en nuestro caso usaremos este servicio tanto para las validaciones de los usuarios como para las validaciones de los tokens por parte de los servicios.

Para la autorización usaremos una infraestructura JWT (<https://jwt.io/introduction>). JSON Web Token (JWT) es un estándar abierto (RFC 7519) que define una forma compacta y autónoma de transmitir información de forma segura entre las partes como un objeto JSON. Esta información se puede verificar y confiar porque está firmada digitalmente. Los JWT se pueden firmar usando una clave secreta (con el algoritmo HMAC) o un par de claves pública/privada usando RSA o ECDSA.

Aunque los JWT se pueden cifrar para proporcionar también confidencialidad entre las partes, nos centraremos en los tokens firmados. Los tokens firmados pueden verificar la integridad de los datos contenidos en ellos, mientras que los tokens encriptados ocultan esos datos de otras partes. Cuando los tokens se firman utilizando pares de claves pública/privada, la firma también certifica que solo la parte que posee la clave privada es la que la firmó.

En caso de una petición no autorizada, devolveremos un código http 401.

### Diseño del API

El API es el punto de entrada al ms, por lo que debe ser cuidadosamente diseñada. En nuestro caso usaremos solamente los métodos GET y POST del protocolo HTTP. Para hacer un diseño adecuado, hay que tener en cuenta que cada petición debe ser completa, es decir, recogerá todos los datos necesarios para realizarla y no dependerá de ningún otro recurso.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Endpoint** | **Necesita token** | **Devuelve** |
| GET | /api/v1/login/?user=X&pass=X | No | Token o 401 si error |
|  | Devolverá un token válido por un día si el usuario y la clave coinciden con los almacenados en la base de datos. | | |
| GET | /api/v1/logout/?token=X | Si | Logout correcto o 401 si error |
|  | Borrará el token correspondiente, invalidándolo para futuras operaciones. | | |
| GET | /api/v1/validate/?token=X | Si | Validation correcto o 401 si error |
|  | Este punto de entrada validará un token, será utilizado principalmente por otros ms para ejecutar las operaciones que se requieran. | | |
| POST | /admin/login/?user=X&pass=X | No | Token o 401 si error |
|  | Al igual que el punto de entada de usuario, pero para gestión de administración. | | |
| POST | /admin/create/usuario=X&pass=X&token=X | Si |  |
|  | Creará un nuevo usuario en la base de datos. | | |
| POST | /admin/delete/usuario=X&token=X | Si |  |
|  | Borrará un usuario de la base de datos. | | |

## Diseño ms rentrada

|  |  |
| --- | --- |
| **Nombre** | rentrada |
| **Dirección** | localhost |
| **Puerto** | 15001 |

Este ms se va a encargar exclusivamente del registro de entrada del instituto. Como el ms de validación necesitará una base de datos propia y unas tablas de gestión desarrolladas en el análisis. El funcionamiento de este ms es simple, todo Endpoint necesitará obligatoriamente un token adquirido en el ms de validación, este ms validará el token contra el ms de validación y si es correcto ejecutará la acción pedida.

### Diseño del API

A continuación, se detallan todos los puntos de entrada de este ms, con estos daremos servicios a todas las operaciones de gestión de nuestro registro de entrada.

El diseño se ha creado en función del análisis para dar respuesta sencilla a las necesidades y que el aprendizaje del ms fuera lo más fácil posible, en un entorno real se deberían tener en cuenta otras consideraciones tales como la seguridad o limitar en número de resultados para evitar la saturación.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Endpoint** | **Necesita autorización** | **Devuelve** |
| GET | /api/v1/get\_rows/?token=X | Si | Json data o 401 si error |
|  | Devuelve todas las líneas del registro de entrada. Hay que tener en cuenta que no van a ser muchas líneas, a lo sumo 200, con lo que no se limita, se deja al cliente. | | |
| GET | /api/v1/get\_rows/<id>/?token=X | Si | Json data o 401 si error |
|  | Con este punto de entrada podremos conseguir todos los datos de una línea del registro. | | |
| GET | /api/v1/insert\_row/?token=X&data=… | Si | Correcto o 401 si error |
|  | Insertamos una línea en el registro con todos los datos, no será necesario que se indique el valor de la clave principal, se creará de forma automática. | | |
| GET | /api/v1/update\_row/id/?token=X&data=… | Si | Correcto o 401 si error |
|  | Se modificarán los datos de una línea de registro, es necesaria la clave primaria para identificar el objeto de los cambios. | | |
| GET | /api/v1/get\_destinos/?token=X | Si | Correcto o 401 si error |
|  | Devuelve todos los datos de la tabla destinos. | | |
| GET | /api/v1/get\_asuntos/?token=X | Si | Correcto o 401 si error |
|  | Devuelve todos los datos de la tabla asuntos. | | |

## Diseño ms rsalida

El diseño de este ms es similar al de entrada con la única diferencia de las tablas a utilizar y los datos a usar (ver la sección de análisis).

|  |  |
| --- | --- |
| **Nombre** | rsalida |
| **Dirección** | localhost |
| **Puerto** | 15002 |

### Diseño del API

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Endpoint** | **Necesita autorización** | **Devuelve** |
| GET | /api/v1/get\_rows/?token=X | Si | Json data o 401 si error |
| GET | /api/v1/get\_rows/<id>/?token=X | Si | Json data o 401 si error |
| GET | /api/v1/insert\_row/?token=X&data=… | Si | Correcto o 401 si error |
| GET | /api/v1/update\_row/id/?token=X&data=… | Si | Correcto o 401 si error |
| GET | /api/v1/get\_destinos/?token=X | Si | Correcto o 401 si error |

## Diseño ms ui usuario

|  |  |
| --- | --- |
| **Nombre** | ui |
| **Dirección** | localhost |
| **Puerto** | 8080 |

Cuando creamos una aplicación se deben gestionar todos los aspectos de la misma. En este caso vamos a crear el interfaz que el usuario utilizará para gestionar ambos registros, de entrada y salida como ejemplo de uso de los ms. No se pretende con este ms hacer una aplicación exhaustiva, simplemente un ejemplo de uso.

Para desarrollar el cliente utilizaremos html/javascript como lenguaje, se creará bajo Flask.

### Diseño del API

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Endpoint** | **Necesita autorización** | **Devuelve** |
|  | /login.html | No |  |
|  | /logout.html | Si |  |
|  | /exportar.html | Si |  |
|  | /rentrada/listado.html | Si |  |
|  | /rentrada/insertar.html | Si |  |
|  | /rentrada/modificar.html | Si |  |
|  | /rsalida/listado.html | Si |  |
|  | /rsalida/insertar.html | Si |  |
|  | /rsalida/modificar.html | Si |  |

## Diseño ms ui administrador

|  |  |
| --- | --- |
| **Nombre** | admin |
| **Dirección** | localhost |
| **Puerto** | 8081 |

Este servicio queda como trabajo al alumno, hay que tener en cuenta que debe ser capaz de crear y borrar nuevos usuarios en el servicio de autorización, para ello tomará primero un token administrador cuando se valide y llamará a las apis correspondientes.

# Capítulo VI Implementación

## Implementación de la BBDD

Usaremos un servidor MySQl o MariaDB que deberá tener una IP fija para que los ms puedan acceder fácilmente, y además permitirá peticiones desde la interfaz externa del host (192.168.91.130). Por defecto se configura para que escuche exclusivamente en la dirección 127.0.0.1.

# En /etc/mysql/mysqld.conf.d/mysqld.cnf

# bind-address = 127.0.0.1,**192.168.91.130**

# mysqlx-bind-address = 127.0.0.1,**192.168.91.130**

También configuraremos la contraseña para que use un encriptado nativo en el usuario que utilizaremos para las conexiones al servidor, no admite otro tipo de cifrado SQLAlchemy.

# Las contraseñas de BBDD en formato nativo obligatoriamente

# ALTER USER '**yourusername**'@'**%**'

IDENTIFIED WITH mysql\_native\_password BY '**youpassword**';

El diseño incluirá tres BBDD diferentes con las siguientes tablas cada una. Las BBDD y los datos necesarios serán los siguientes.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Nombre BBDD** | **Tabla** | **Usuario** | **Clave** | **Permisos** | **Ms relacionado** |
| autorizacion | logins | log | log | Insert/Delete/Select | Autorización |
|  |  | root | root | Control Total |  |
|  | | | | | |
| rentrada | registro | log | log | Insert/Delete/Select/Update | rentrada |
|  | easunto | log | log | Select | rentrada |
|  | edestino | log | log | Select | rentrada |
|  |  | root | root | Control Total |  |
|  | | | | | |
| rsalida | registro | log | log | Insert/Delete/Select/Update | rsalida |
|  | sdestino | log | log | Select | rsalida |
|  |  | root | root | Control Total |  |

### Scripts de creación

#### Autorización

-- phpMyAdmin SQL Dump

-- version 5.0.4deb2ubuntu5

-- https://www.phpmyadmin.net/

--

-- Servidor: localhost:3306

-- Tiempo de generación: 17-03-2022 a las 17:45:33

-- Versión del servidor: 8.0.28-0ubuntu0.21.10.3

-- Versión de PHP: 8.0.8

SET SQL\_MODE = "NO\_AUTO\_VALUE\_ON\_ZERO";

START TRANSACTION;

SET time\_zone = "+00:00";

/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET @OLD\_CHARACTER\_SET\_RESULTS=@@CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET @OLD\_COLLATION\_CONNECTION=@@COLLATION\_CONNECTION \*/;

/\*!40101 SET NAMES utf8mb4 \*/;

--

-- Base de datos: `autorizacion`

--

CREATE DATABASE IF NOT EXISTS `autorizacion` DEFAULT CHARACTER SET utf8 COLLATE utf8\_spanish2\_ci;

USE `autorizacion`;

-- --------------------------------------------------------

--

-- Estructura de tabla para la tabla `logins`

--

DROP TABLE IF EXISTS `logins`;

CREATE TABLE IF NOT EXISTS `logins` (

`login` varchar(255) COLLATE utf8\_spanish2\_ci NOT NULL,

`pasw` varchar(255) COLLATE utf8\_spanish2\_ci NOT NULL,

`tipo` varchar(1) COLLATE utf8\_spanish2\_ci NOT NULL DEFAULT 'U' COMMENT 'A --> admin, U --> User',

`token` text CHARACTER SET utf8 COLLATE utf8\_spanish2\_ci,

PRIMARY KEY (`login`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3 COLLATE=utf8\_spanish2\_ci;

--

-- Volcado de datos para la tabla `logins`

--

INSERT INTO `logins` (`login`, `pass`, `token`, `fecha\_fin\_valido`) VALUES

('a', 'a', 'A', NULL, NULL),

('u', 'u', 'U', NULL, NULL);

COMMIT;

/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET CHARACTER\_SET\_RESULTS=@OLD\_CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET COLLATION\_CONNECTION=@OLD\_COLLATION\_CONNECTION \*/;

#### rentrada

-- phpMyAdmin SQL Dump

-- version 5.0.4deb2ubuntu5

-- https://www.phpmyadmin.net/

--

-- Servidor: localhost:3306

-- Tiempo de generación: 17-03-2022 a las 17:47:00

-- Versión del servidor: 8.0.28-0ubuntu0.21.10.3

-- Versión de PHP: 8.0.8

SET SQL\_MODE = "NO\_AUTO\_VALUE\_ON\_ZERO";

START TRANSACTION;

SET time\_zone = "+00:00";

/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET @OLD\_CHARACTER\_SET\_RESULTS=@@CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET @OLD\_COLLATION\_CONNECTION=@@COLLATION\_CONNECTION \*/;

/\*!40101 SET NAMES utf8mb4 \*/;

--

-- Base de datos: `rentrada`

--

CREATE DATABASE IF NOT EXISTS `rentrada` DEFAULT CHARACTER SET utf8 COLLATE utf8\_spanish2\_ci;

USE `rentrada`;

-- --------------------------------------------------------

--

-- Estructura de tabla para la tabla `easunto`

--

DROP TABLE IF EXISTS `easunto`;

CREATE TABLE IF NOT EXISTS `easunto` (

`id` int NOT NULL,

`descri` varchar(50) COLLATE utf8\_spanish2\_ci NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3 COLLATE=utf8\_spanish2\_ci;

--

-- Volcado de datos para la tabla `easunto`

--

INSERT INTO `easunto` (`id`, `descri`) VALUES

(1, 'Concurso de Traslados'),

(2, 'Oposiciones'),

(3, 'Otro');

-- --------------------------------------------------------

--

-- Estructura de tabla para la tabla `edestino`

--

DROP TABLE IF EXISTS `edestino`;

CREATE TABLE IF NOT EXISTS `edestino` (

`id` int NOT NULL,

`descri` varchar(50) COLLATE utf8\_spanish2\_ci NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3 COLLATE=utf8\_spanish2\_ci;

--

-- Volcado de datos para la tabla `edestino`

--

INSERT INTO `edestino` (`id`, `descri`) VALUES

(1, 'Director'),

(2, 'Otro');

-- --------------------------------------------------------

--

-- Estructura de tabla para la tabla `registro`

--

DROP TABLE IF EXISTS `registro`;

CREATE TABLE IF NOT EXISTS `registro` (

`referencia` int NOT NULL,

`fecha` varchar(25) COLLATE utf8\_spanish2\_ci NOT NULL,

`asunto` int NOT NULL,

`asuntootro` varchar(50) CHARACTER SET utf8 COLLATE utf8\_spanish2\_ci DEFAULT NULL,

`destino` int NOT NULL,

`destinootro` varchar(50) DEFAULT NULL,

`interesado` varchar(50) NOT NULL,

`resumen` varchar(100) DEFAULT NULL,

`observaciones` mediumtext CHARACTER SET utf8 COLLATE utf8\_spanish2\_ci,

`fecham` varchar(25) CHARACTER SET utf8 COLLATE utf8\_spanish2\_ci DEFAULT NULL,

`mmod` varchar(100) CHARACTER SET utf8 COLLATE utf8\_spanish2\_ci DEFAULT NULL,

PRIMARY KEY (`referencia`),

KEY `asunto` (`asunto`),

KEY `destino` (`destino`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3 COLLATE=utf8\_spanish2\_ci;

--

-- Restricciones para tablas volcadas

--

--

-- Filtros para la tabla `registro`

--

ALTER TABLE `registro`

ADD CONSTRAINT `registro\_ibfk\_1` FOREIGN KEY (`destino`) REFERENCES `edestino` (`id`) ON DELETE CASCADE ON UPDATE CASCADE,

ADD CONSTRAINT `registro\_ibfk\_2` FOREIGN KEY (`asunto`) REFERENCES `easunto` (`id`) ON DELETE CASCADE ON UPDATE CASCADE;

COMMIT;

/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET CHARACTER\_SET\_RESULTS=@OLD\_CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET COLLATION\_CONNECTION=@OLD\_COLLATION\_CONNECTION \*/;

#### rsalida

-- phpMyAdmin SQL Dump

-- version 5.0.4deb2ubuntu5

-- https://www.phpmyadmin.net/

--

-- Servidor: localhost:3306

-- Tiempo de generación: 17-03-2022 a las 17:54:25

-- Versión del servidor: 8.0.28-0ubuntu0.21.10.3

-- Versión de PHP: 8.0.8

SET SQL\_MODE = "NO\_AUTO\_VALUE\_ON\_ZERO";

START TRANSACTION;

SET time\_zone = "+00:00";

/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET @OLD\_CHARACTER\_SET\_RESULTS=@@CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET @OLD\_COLLATION\_CONNECTION=@@COLLATION\_CONNECTION \*/;

/\*!40101 SET NAMES utf8mb4 \*/;

--

-- Base de datos: `rsalida`

--

CREATE DATABASE IF NOT EXISTS `rsalida` DEFAULT CHARACTER SET utf8 COLLATE utf8\_spanish2\_ci;

USE `rsalida`;

-- --------------------------------------------------------

--

-- Estructura de tabla para la tabla `registro`

--

DROP TABLE IF EXISTS `registro`;

CREATE TABLE IF NOT EXISTS `registro` (

`referencia` int NOT NULL,

`fecha` varchar(25) CHARACTER SET utf8 COLLATE utf8\_spanish2\_ci NOT NULL,

`destino` int NOT NULL,

`destinootro` varchar(50) DEFAULT NULL,

`interesado` varchar(50) NOT NULL,

`observaciones` mediumtext CHARACTER SET utf8 COLLATE utf8\_spanish2\_ci,

`fecham` varchar(25) CHARACTER SET utf8 COLLATE utf8\_spanish2\_ci DEFAULT NULL,

`mmod` varchar(100) CHARACTER SET utf8 COLLATE utf8\_spanish2\_ci DEFAULT NULL,

PRIMARY KEY (`referencia`),

KEY `destino` (`destino`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3 COLLATE=utf8\_spanish2\_ci;

-- --------------------------------------------------------

--

-- Estructura de tabla para la tabla `sdestino`

--

DROP TABLE IF EXISTS `sdestino`;

CREATE TABLE IF NOT EXISTS `sdestino` (

`id` int NOT NULL,

`descri` varchar(50) CHARACTER SET utf8 COLLATE utf8\_spanish2\_ci NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3 COLLATE=utf8\_spanish2\_ci;

--

-- Volcado de datos para la tabla `sdestino`

--

INSERT INTO `sdestino` (`id`, `descri`) VALUES

(1, 'Consejería de Educación y Ciencia'),

(2, 'Interno'),

(3, 'Otro'),

(4, 'Delegación de Edu. Albacete'),

(5, 'Delegación de Edu. Ciudad Real'),

(6, 'Delegación de Edu. Cuenca'),

(7, 'Delegación de Edu. Guadalaja'),

(8, 'Delegación de Edu. Toledo');

--

-- Restricciones para tablas volcadas

--

--

-- Filtros para la tabla `registro`

--

ALTER TABLE `registro`

ADD CONSTRAINT `registro\_ibfk\_1` FOREIGN KEY (`destino`) REFERENCES `sdestino` (`id`) ON DELETE CASCADE ON UPDATE CASCADE;

COMMIT;

/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET CHARACTER\_SET\_RESULTS=@OLD\_CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET COLLATION\_CONNECTION=@OLD\_COLLATION\_CONNECTION \*/;

## Servicio de autorización

La implementación del servicio de autorización es la base para el desarrollo posterior del sistema basado en ms. En concreto sentará las bases de programación y administración que utilizarán el resto de servicios.

***Reflexión personal.***

*Desarrollar este servicio me ha llevado más tiempo del esperado, es imprescindible dominar los conceptos de redes, Docker y sistemas operativos para llevar a cabo el desarrollo. La creación del servicio en sí y sus pruebas no ha sido excesivamente laborioso, pero “dockerizar” la aplicación ha costado bastante.*

*En concreto, partimos de una imagen Docker 3.9 de Python similar al entorno de desarrollo, pero las librerías que se instalaban para poder compilar la librería mysqlclient para Python no se correspondían con la versión 3.9, eran la de la versión base 3.7, con lo que se cambió la imagen a 3.7-python y se solucionaron los problemas de conexión con el servidor externo.*

*Otro de los problemas que me he encontrado es que sin dominar Docker no se puede crear una aplicación basada en ms. El entender el funcionamiento de las redes y los comandos básicos de log son imprescindibles.*

*Así mismo, es necesario conocer la configuración del servicio que se instala en el contenedor, en este caso uwsgi, procedimiento de personal de administración de sistemas.*

*En resumen, el número de tecnologías y conocimientos necesarios son mucho más numerosos que los necesarios para el desarrollo de una aplicación monolítica*

Antes de comenzar recordaremos algunos conceptos importantes:

**WSGI** (Interfaz de puerta de enlace del servidor web) es un protocolo estandarizado que define como un servidor web interactúa con las aplicaciones, en nuestro caso la aplicación Python. El punto de entrada a la aplicación es el fichero **wsgi.py** que será ejecutado en el momento de lanzar el servidor.

**UWSGI** es un protocolo binario que puede transportar cualquier tipo de datos entre un servidor web y la aplicación con la que está interactuando.

**uWSGI** es un servidor web ligero para producción que implementa los protocolos uwsgi y WSGI. En este microservicio utilizaremos uWsgi como servidor web para ejecutar nuestra aplicación. Dicha aplicación implementará un conjunto de puntos de entrada a través del protocolo HTTP y los comandos GET y POST tal y como se definieron en el análisis.

### wsgi.py

Este fichero contiene la inicialización de nuestra aplicación, será ejecutado por el servidor web en el momento del arranque. Tiene muy poco contenido, los *imports* necesarios de configuración y la llamada a la función de inicialización de nuestra aplicación.

Servidor

Web

Wsgi.py

App.py

from auth.app import create\_app

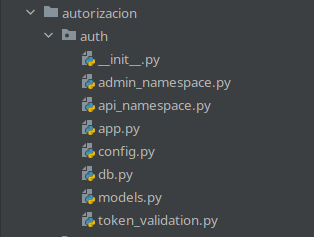
from auth.config import SRV\_PORT, SRV\_IP

application = create\_app()

if \_\_name\_\_ == '\_\_main\_\_': **# si se ejecuta desde la consola directamente**

application.run(host=SRV\_IP, port=SRV\_PORT)

### Directorio auth

El directorio *auth* contendrá todo el código fuente de nuestro ms. Se va a dividir es siete ficheros:

* **config.py** contendrá los valores de configuración de la aplicación.
* **db.py** define la clase de acceso a la tabla logins de la base de datos autorización, así como las configuraciones necesarias para conectarnos al servidor MariaDB o MySQL.
* **models.py** tiene las estructuras de datos necesarias para las comunicaciones, en concreto el modelo de acceso a la base de datos, la estructura de datos que contendrá nuestro token de autorización y la estructura de datos del mensaje entre el cliente y el servidor.
* **app.py** es el fichero base de nuestra aplicación, define la función de creación de la aplicación. Esta se basa en el Frameworks flask\_resk.
* **token\_validation.py** agrupa todas las funciones para la gestión del token, creación, validación, etc.
* **api\_namespace.py** define los puntos de entrada y valores a pasar definidos en el análisis de ms para el usuario.
* **admin\_namespace.py** tiene todos los puntos de entrada y valores a pasar definidos en el análisis del ms para el administrador.

#### config.py

El fichero de configuración contiene las claves RSA privadas y públicas (**PRIVATE\_KEY, PUBLIC\_KEY**) que utilizaremos para la creación y validación del token que el usuario va a utilizar, se pueden generar a través de herramientas locales o conseguir de un proveedor, en cualquier caso, ambas son necesarias. Si llevamos la aplicación a un entorno de producción deberán ser siempre de un proveedor acreditado, no podemos usar claves generadas por nosotros.

Además, establece tres variables de gestión: una indica el tiempo que el token será válido (**TIME\_VALID\_TOKEN**, 3600 segundos), la utilizaremos en el momento que creemos un token para el usuario o administrador tras la validación y en la verificación del mismo.

Las otras dos constantes indican la ip y el puerto (**SRV\_IP, SRV\_PORT**) de nuestra aplicación. Estos datos se recogen de variables del entorno del SSOO. Si el SSOO no tiene definida las variables se establecen los valores por defecto. Este mecanismo nos va a permitir ejecutar nuestra aplicación en un entorno Docker y establecer estas variables en el momento de la creación del contenedor (*run*) o mediante las opciones en la ejecución Docker-compose o kubernets.

import os

PRIVATE\_KEY = '''

-----BEGIN RSA PRIVATE KEY-----

-----END RSA PRIVATE KEY-----

'''

PUBLIC\_KEY = '''

-----BEGIN PUBLIC KEY-----

-----END PUBLIC KEY-----

'''

# Tiempo en minutos de validación del token

TIME\_VALID\_TOKEN = 3600

SRV\_IP = os.environ.get('SRV\_IP', '127.0.0.1')

SRV\_PORT = os.environ.get('SRV\_PORT', '15000')

#### db.py

El fichero de gestión de la BBDD contiene un conjunto de constantes que se establecerán, del mismo modo que en el fichero anterior a través de variables de entorno, para el acceso y la gestión del servidor de la Base de datos (**DB\_XXXX**).

A continuación, crea la configuración (**db\_config**) para el acceso a la base de datos y el objeto que se utilizará para la gestión de la misma dentro de nuestra aplicación (**db**).

Por último, se crea la clase que identifica la tabla logins (Logins) del servidor, es utilizada por la librería SQLAchemy para poder gestionar las órdenes SQL de gestión (INSERT, SELECT, etc) a través del objeto db. La unión se hace a través de la herencia: **class Logins(db.Model)**.

import os

from flask\_sqlalchemy import SQLAlchemy

DB\_USER = os.environ.get('DB\_USER', 'log')

DB\_PW = os.environ.get('DB\_PW', 'log')

DB\_SRV = os.environ.get('DB\_SRV', 'localhost')

DB\_BBDD = os.environ.get('DB\_BBDD', 'autorizacion')

DB\_URI = f"mysql://{DB\_USER}:{DB\_PW}@{DB\_SRV}/{DB\_BBDD}"

db\_config = {

'SQLALCHEMY\_DATABASE\_URI': DB\_URI,

'SQLALCHEMY\_TRACK\_MODIFICATIONS': False,

}

db = SQLAlchemy()

class Logins(db.Model):

USUARIO = 'U'

ADMIN = 'A'

login = db.Column(db.String(255), primary\_key=True)

pasw = db.Column(db.String(255), nullable=False)

tipo = db.Column(db.String(1), default=USUARIO)

token = db.Column(db.Text)

#### models.py

El fichero models va a implementar las estructuras de datos de gestión de nuestra aplicación. De este fichero son relevantes los payload. El **token\_payload** se utiliza para la generación del token JWT para la gestión de la autorización, son los campos necesarios para una correcta generación y posterior validación. Por otro lado, **return\_payload** contiene la estructura de campos que se va a utilizar en la respuesta al cliente en cualquiera de las peticiones al servidor.

from flask\_restx import fields

login\_model = {

'login': fields.Integer(),

'pasw': fields.String(),

'tipo': fields.String(),

'token': fields.String(),

}

token\_payload = {

'login': fields.String(),

'tipo': fields.String(),

'iat': fields.DateTime(),

'exp': fields.DateTime()

}

return\_payload = {

'message': fields.String(),

}

#### app.py

El fichero principal de creación de nuestra aplicación. Simplemente exporta una función para crear las estructuras flask de la misma. Esta función es llamada en el momento de la primera ejecución de nuestro servidor uwsgi. Cuando un usuario acceda a una url del servidor se buscará a través de los puntos de acceso registrados en los ficheros api\_namespace y admin\_namespace y si hay coincidencia se ejecutará la función asignada.

from flask import Flask

from flask\_restx import Api

from auth.api\_namespace import api\_namespace

from auth.admin\_namespace import admin\_namespace

from auth.db import db, db\_config

# import logging

# logger = logging.getLogger(\_\_name\_\_)

# logger.info('App init') # Esta línea en cualquier parte del código para mandar al log

def create\_app():

application = Flask(\_\_name\_\_) **# Usaremos el Framework Flask para la gestión**

api = Api(application, version='0.1', title='Autorizacion ms', description='ms de autorización')

application.config['RESTPLUS\_MASK\_SWAGGER'] = False

application.config.update(db\_config)

db.init\_app(application)

application.db = **db # Creamos un objeto interno a la aplicación para acceso a la BBDD**

api.add\_namespace(api\_namespace) **# Registramos el api de usuario**

api.add\_namespace(admin\_namespace) **# Registramos el api del administrador**

return application

#### token\_validation.py

Recoge todas las funciones para la creación y validación del token jwt (ver capítulos anteriores) que se mandará al usuario. Para recordar el funcionamiento leer la sección de diseño.

import jwt

from datetime import datetime, timedelta

from http import HTTPStatus

from auth.db import db, Logins

from auth.models import token\_payload

from auth.config import TIME\_VALID\_TOKEN

Las dos siguientes funciones se encargan de codificar y decodificar un token, usando las claves privadas y públicas correspondientemente. Usan un algoritmo RSA 256 bits para la codificación.

def encode\_token(data, private\_key):

return jwt.encode(data, private\_key, algorithm='RS256')

def decode\_token(token, public\_key):

return jwt.decode(token, public\_key, algorithms='RS256')

Crea un token para un usuario si conoce la clave y el nombre del usuario, tendrá una validez de un día. La función preguntará a la base de datos si el usuario y clave están almacenados y en caso afirmativo crea un nuevo token y lo devuelve, así como un código de error.

Los datos necesarios para crear el token son el nombre del usuario (*login*), el *tipo* (administrador o usuario), el campo *iat* que define el momento en el que se crea el token y el camp *exp* que define el momento en el que expira el token. Los campos iat y exp son obligatorios para que la librearía compruebe automáticamente la validez del token y no tenerlo que hacer de forma manual.

El token queda almacenado en la base de datos, en el registro del usuario, en el correspondiente campo para validaciones posteriores si fuera necesario.

def generate\_token(usuario, password, private\_key):

"""

:param usuario:

:param password:

:param private\_key:

:return: http FORBIDDEN| OK, token|None

"""

usuario\_bbdd = Logins.query.filter\_by(login=usuario, pasw=password).first()

if usuario\_bbdd: **# si el usuario valida se crea el token y se codifica**

token\_payload['login'] = usuario\_bbdd.login

token\_payload['tipo'] = usuario\_bbdd.tipo

token\_payload['iat'] = datetime.utcnow()

token\_payload['exp'] = datetime.utcnow() + timedelta(minutes=TIME\_VALID\_TOKEN)

token = encode\_token(token\_payload, private\_key)

usuario\_bbdd.token = token **# almacenar token en la base de datos**

db.session.commit()

return HTTPStatus.OK, f'{token}'

return HTTPStatus.FORBIDDEN, None

La siguiente función admite el último token creado por el usuario. Para poder validar el token, primero hay que decodificarlo con la clave pública del usuario. De esta manera se evita que el usuario pueda cambiarlo o corromperlo.

Tras la decodificación y un conjunto de comprobaciones de error que se pueden dar en el token, se pasa a aceptar que el token del usuario comprobando si el almacenado en la base de datos es el mismo que el que nos ha llegado, autorizando en caso afirmativo. En el resto de opciones se generará un error.

def validate\_token(token, public\_key):

"""

:param token:

:param public\_key:

:return: http METHOD\_NOT\_ALLOWED| OK, login|None

"""

try:

decoded\_token = decode\_token(token, public\_key)

except jwt.exceptions.DecodeError:

return HTTPStatus.METHOD\_NOT\_ALLOWED, None

except jwt.exceptions.ExpiredSignatureError:

return HTTPStatus.METHOD\_NOT\_ALLOWED, None

if 'exp' not in decoded\_token:

return HTTPStatus.METHOD\_NOT\_ALLOWED, None

if 'login' not in decoded\_token:

return HTTPStatus.METHOD\_NOT\_ALLOWED, None

if 'iat' not in decoded\_token:

return HTTPStatus.METHOD\_NOT\_ALLOWED, None

if 'tipo' not in decoded\_token:

return HTTPStatus.METHOD\_NOT\_ALLOWED, None

usuario\_bbdd = Logins.query.filter\_by(login=decoded\_token['login']).first()

if not usuario\_bbdd or token != usuario\_bbdd.token:

return HTTPStatus.METHOD\_NOT\_ALLOWED, None

return HTTPStatus.OK, decoded\_token['login']

logout\_token borra el token del usuario almacenado en la base de datos.

def logout\_token(usuario):

"""

:param usuario:

:return: http FORBIDDEN|OK, usuario|None

"""

usuario\_bbdd = Logins.query.filter\_by(login=usuario).first()

if usuario\_bbdd:

usuario\_bbdd.token = '' **# Borra el token de la bbdd**

db.session.commit()

return HTTPStatus.OK, usuario

return HTTPStatus.FORBIDDEN, None

generate\_token\_admin crea un token para un administrador si conoce la clave y el nombre del usuario, tendrá una validez de un día.

def generate\_token\_admin(usuario, password, private\_key):

"""

:param usuario:

:param password:

:param private\_key:

:return: http FORBIDDEN| OK, token|None

"""

usuario\_bbdd = Logins.query.filter\_by(login=usuario, pasw=password).first()

if usuario\_bbdd and usuario\_bbdd.tipo == Logins.ADMIN:

token\_payload['login'] = usuario\_bbdd.login

token\_payload['tipo'] = usuario\_bbdd.tipo

token\_payload['iat'] = datetime.utcnow()

token\_payload['exp'] = datetime.utcnow() + timedelta(minutes=TIME\_VALID\_TOKEN)

token = encode\_token(token\_payload, private\_key)

usuario\_bbdd.token = token

db.session.commit()

return HTTPStatus.OK, f'{token}'

return HTTPStatus.FORBIDDEN, None

A continuación se valida el último token creado por el administrador.

def validate\_token\_admin(token, public\_key):

"""

:param token:

:param public\_key:

:return: http METHOD\_NOT\_ALLOWED| OK, login|None

"""

try:

decoded\_token = decode\_token(token, public\_key)

except jwt.exceptions.DecodeError:

return HTTPStatus.METHOD\_NOT\_ALLOWED, None

except jwt.exceptions.ExpiredSignatureError:

return HTTPStatus.METHOD\_NOT\_ALLOWED, None

if 'exp' not in decoded\_token:

return HTTPStatus.METHOD\_NOT\_ALLOWED, None

if 'login' not in decoded\_token:

return HTTPStatus.METHOD\_NOT\_ALLOWED, None

if 'iat' not in decoded\_token:

return HTTPStatus.METHOD\_NOT\_ALLOWED, None

if 'tipo' not in decoded\_token:

return HTTPStatus.METHOD\_NOT\_ALLOWED, None

usuario\_bbdd = Logins.query.filter\_by(login=decoded\_token['login'], tipo=Logins.ADMIN).first()

if not usuario\_bbdd or token != usuario\_bbdd.token:

return HTTPStatus.METHOD\_NOT\_ALLOWED, None

return HTTPStatus.OK, decoded\_token['login']

#### api\_namespace.py

Registra las url de acceso por el servidor web del usuario. Una de las bases de los servicios es que estén bien documentados, y el mantenimiento de la documentación se hace muy tediosos. Si incluimos en la programación dicha documentación aumentaremos la productividad y minimizaremos los errores. En nuestro caso, las llamadas a *@....doc*, *@....response* son llamadas de documentación. Para acceder a la documentación generada automáticamente hay que usar el cliente web y utilizar la url de acceso definida en el registro (http://nombre\_servidor/api/).

from http import HTTPStatus

from flask\_restx import Namespace, Resource, abort

from flask import request

from auth.config import PUBLIC\_KEY, PRIVATE\_KEY

from auth.models import return\_payload

from auth.token\_validation import validate\_token, generate\_token, logout\_token

api\_namespace = Namespace('api', description='User API operations')

return\_payload\_model = api\_namespace.model('Return', return\_payload)

Definimos la ruta de validación del usuario, necesitará una clave y usuario de entrada y comprobará que es válido. La respuesta del servicio tendrá la forma del *return\_payload* (que recordemos solo tenía un campo). El decorador *@....marshal\_with* hace que se transforme nuestro objeto en contenido json de forma automática antes de ser devuelto al usuario vía http.

@api\_namespace.route('/v1/login/')

class LoginMs(Resource):

@api\_namespace.doc('login')

@api\_namespace.doc(params={'user': 'Nombre de usuario', 'pass': 'Clave'})

@api\_namespace.response(200, 'Token creado', return\_payload\_model)

@api\_namespace.response(403, 'Error de acceso', return\_payload\_model)

@api\_namespace.marshal\_with(return\_payload\_model)

def get(self):

us = request.args.get("user", None)

pw = request.args.get("pass", None)

code, token = generate\_token(us, pw, PRIVATE\_KEY)

if code == HTTPStatus.OK:

return\_payload['message'] = token

return return\_payload

abort(code)

Definimos la ruta para validar un token, necesitará un token de entrada y comprobará que es válido. Este punto de acceso del ms será utilizado generalmente por otros ms para determinar si un token todavía es válido y pueden llevar a cabo la acción solicitada

@api\_namespace.route('/v1/validate/')

class ValidateMs(Resource):

@api\_namespace.doc('validate')

@api\_namespace.doc(params={'token': 'Token de validación'})

@api\_namespace.response(200, 'Validacion correcta', return\_payload\_model)

@api\_namespace.response(403, 'Error de acceso', return\_payload\_model)

@api\_namespace.response(405, 'Error de token', return\_payload\_model)

@api\_namespace.marshal\_with(return\_payload\_model)

def get(self):

tk = request.args.get("token", None) **# token enviado por el usuario**

code, login = validate\_token(tk, PUBLIC\_KEY) **# ¿token válido?**

if code == HTTPStatus.OK:

return\_payload['message'] = True

return return\_payload

abort(code)

Ruta de acceso para la desconexión. Este paso no es obligatorio ya que el token generado tiene una validez de un día, con lo que si no se desconecta tras ese tiempo no será accesible. Pero si tiene implicaciones de seguridad, ya que si no se desconecta y es capturado el token, cualquiera podría actuar en su nombre durante la validez del mismo.

@api\_namespace.route('/v1/logout/')

class LogoutMs(Resource):

@api\_namespace.doc('logout')

@api\_namespace.doc(params={'token': 'token de validación'})

@api\_namespace.response(200, 'Logout correcta', return\_payload\_model)

@api\_namespace.response(403, 'Error de acceso', return\_payload\_model)

@api\_namespace.response(405, 'Error de token', return\_payload\_model)

@api\_namespace.marshal\_with(return\_payload\_model)

def get(self):

tk = request.args.get("token", None) **# token enviado por el usuario**

code, login = validate\_token(tk, PUBLIC\_KEY) **# ¿token válido?**

if code == HTTPStatus.OK:

logout\_token(login)

return\_payload['message'] = True

return return\_payload

abort(code)

#### admin\_namespace.py

Similar a lo visto en el punto anterior para el usuario, pero en este caso para el administrador.

from http import HTTPStatus

from flask\_restx import Namespace, Resource, abort

from flask import request

from auth.config import PUBLIC\_KEY, PRIVATE\_KEY

from auth.models import return\_payload

from auth.token\_validation import generate\_token\_admin, validate\_token\_admin

from auth.db import db, Logins

admin\_namespace = Namespace('admin', description='Admin API operations')

return\_payload\_model = admin\_namespace.model('Return', return\_payload)

Similar a lo visto en el punto anterior para el usuario, excepto que solo usará POST.

@admin\_namespace.route('/login/', methods=['POST'])

class LoginAdminMs(Resource):

@admin\_namespace.doc('login')

@admin\_namespace.doc(params={'user': 'Nombre de usuario', 'pass': 'Clave'})

@admin\_namespace.response(200, 'Token creado', return\_payload\_model)

@admin\_namespace.response(403, 'Error de acceso', return\_payload\_model)

@admin\_namespace.marshal\_with(return\_payload\_model)

def post(self):

us = request.form.get("user", None)

pw = request.form.get("pass", None)

code, token = generate\_token\_admin(us, pw, PRIVATE\_KEY)

if code == HTTPStatus.OK:

return\_payload['message'] = token

return return\_payload

abort(code)

Similar a lo visto en el punto anterior para el usuario, excepto que solo usará POST. Este punto crea un nuevo usuario tipo *usuario* en el sistema.

@admin\_namespace.route('/create/', methods=['POST'])

class CreateAdminMs(Resource):

@admin\_namespace.doc('create')

@admin\_namespace.doc(params={'user': 'Nombre de usuario', 'pass': 'Clave', 'token': 'token'})

@admin\_namespace.response(200, 'Usuario creado', return\_payload\_model)

@admin\_namespace.response(403, 'Error de acceso', return\_payload\_model)

@admin\_namespace.response(405, 'Error de token', return\_payload\_model)

@admin\_namespace.marshal\_with(return\_payload\_model)

def post(self):

us = request.form.get("user", None)

pw = request.form.get("pass", None)

tk = request.form.get("token", None)

code, \_ = validate\_token\_admin(tk, PUBLIC\_KEY)

if code == HTTPStatus.OK:

usuario\_bbdd = Logins()

usuario\_bbdd.login = us

usuario\_bbdd.pasw = pw

db.session.add(usuario\_bbdd)

db.session.commit()

return\_payload['message'] = True

return return\_payload

abort(code)

Similar a lo visto en el punto anterior para el usuario, excepto que solo usará POST. Borra un usuario del sistema.

@admin\_namespace.route('/delete/', methods=['POST'])

class DeleteAdminMs(Resource):

@admin\_namespace.doc('delete')

@admin\_namespace.doc(params={'user': 'Nombre de usuario', 'pass': 'Clave', 'token': 'token'})

@admin\_namespace.response(200, 'Usuario borrado', return\_payload\_model)

@admin\_namespace.response(403, 'Error de acceso', return\_payload\_model)

@admin\_namespace.response(405, 'Error de token', return\_payload\_model)

@admin\_namespace.marshal\_with(return\_payload\_model)

def post(self):

us = request.form.get("user", None)

pw = request.form.get("pass", None)

tk = request.form.get("token", None)

code, \_ = validate\_token\_admin(tk, PUBLIC\_KEY)

if code == HTTPStatus.OK:

usuario\_bbdd = Logins.query.filter\_by(login=us, pasw=pw).first()

if usuario\_bbdd:

usuario\_bbdd.login = us

usuario\_bbdd.pasw = pw

db.session.delete(usuario\_bbdd)

db.session.commit()

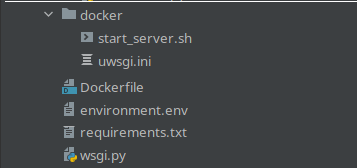
return\_payload['message'] = True

return return\_payload

abort(HTTPStatus.METHOD\_NOT\_ALLOWED)

abort(code)

### Dockerizar el servicio

Si estamos creando ms es porque necesitamos ciertas características de funcionamiento, escalado, etc. El sistema utilizará Docker para el control de los ms. En este apartado vamos a ver el proceso de migrar nuestro ms que funciona en la máquina local a que funcione en Docker.

Para poder implementar este paso debemos generar en primer lugar los requerimientos Python de librerías que tendrá nuestra imagen Docker, así como decidir la base de la misma. Para este ms usaremos la base python:3.7-slim-buster.

#### requeriments.txt

La instrucción siguiente creará un fichero con todas las librerías que son necesarias instalar dentro de la imagen para que funcione nuestra aplicación.

pip3 freeze > requirements.txt

#### environment.env

Este fichero contendrá todas las variables de entorno que queremos crear en el momento de la ejecución de nuestro contenedor, son las mismas que ya se definieron dentro del ms. De esta manera permitimos configuraciones externas a nuestro ms de forma automática.

SRV\_IP=127.0.0.1

SRV\_PORT=15000

DB\_USER=log

DB\_PW=log

DB\_SRV=192.168.91.130

DB\_BBDD=autorizacion

#### Docker/start\_server.sh

Los contenedores deben ejecutar el proceso una vez que arrancar a través de la orden CMD o de la orden ENTRYPOINT, este es el fichero que será ejecutado en ese caso para lanzar el servidor web uwsgi que instalaremos.

Debemos reseñar que el fichero de configuración del servidor reside bajo */opt/code* y que se tiene que lanzar con soporte Python (*--plugin python3*).

#!/bin/sh

\_term() {

echo "Caught SIGTERM signal! Sending graceful stop to uWSGI through the master-fifo"

# See details in the uwsgi.ini file and

# in http://uwsgi-docs.readthedocs.io/en/latest/MasterFIFO.html

# q means "graceful stop"

echo q > /tmp/uwsgi-fifo

}

trap \_term SIGTERM

**uwsgi --plugin python3 --http-socket :$SRV\_PORT --ini /opt/code/docker/uwsgi.ini &**

# We need to wait to properly catch the signal, that's why uWSGI is started

# in the background. $! is the PID of uWSGI

wait $!

# The container exits with code 143, which means "exited because SIGTERM"

# 128 + 15 (SIGTERM)

#### Docker/uwsgi.ini

Fichero de configuración del servidor web, se configura el directorio de la aplicación por defecto (*/opt/code*), el fichero a ejecutar cuando se lance (*wsgi.py*) y el usuario del sistema con el que correrá el servidor (*uid=uwsgi*). Si deseamos conocer el resto de opciones visitar la página de referencia del servidor (https://uwsgi-docs.readthedocs.io/en/latest/).

[uwsgi]

**uid=uwsgi**

**chdir=/opt/code**

**wsgi-file=/opt/code/wsgi.py**

master=True

pidfile=/tmp/uwsgi.pid

socket=/tmp/uwsgi.sock

vacuum=True

processes=1

max-requests=5000

# Used to send commands to uWSGI

master-fifo=/tmp/uwsgi-fifo

plugins=python

pythonpath=/usr/local/lib/python3.7/site-packages/

#### Dockerfile

FROM python:3.7-slim-buster

ENV SRV\_PORT=$SRV\_PORT

ENV SRV\_IP=$SRV\_IP

ENV DB\_USER=$DB\_USER

ENV DB\_PW=$DB\_PW

ENV DB\_SRV=$DB\_SRV

ENV DB\_BBDD=$DB\_BBDD

ENV LD\_LIBRARY\_PATH="/usr/lib/x86\_64-linux-gnu/:$PATH"

Instalamos los paquetes básicos para el funcionamiento del servidor.

RUN apt-get update \

&& apt-get -y install gcc libffi-dev python3-dev \

&& apt-get -y install uwsgi uwsgi-core uwsgi-plugin-python3

Instalamos los paquetes para la creación de la librería cliente mysql de Python, no se puede instalar desde pip y hay que compilarla.

RUN apt-get -y install default-libmysqlclient-dev

Creamos el directorio de la aplicación y nos establecemos en él.

RUN mkdir -p /opt/code

WORKDIR /opt/code

Copiamos las librerías Python que son necesarias y las instalamos en el sistema.

COPY requirements.txt requirements.txt

RUN pip3 install --upgrade pip

RUN pip3 install -r requirements.txt

Creamos un usuario y grupo uwsgi para el servidor por seguridad, no se debe ejecutar bajo root nunca.

RUN addgroup --system uwsgi \

&& adduser --no-create-home --ingroup uwsgi --disabled-password --system uwsgi

Borramos todos los datos innecesarios, este paso no afecta al tamaño de la imagen final.

RUN apt-get clean

RUN rm -rf /varlib/a/list/\*

RUN rm -rf /var/cache/apt/archives/\*

Cambiamos de usuario y copiamos todos los ficheros fuentes en el directorio /opt/code haciendo que pertenezcan al usuario y grupo correcto.-

USER uwsgi

COPY --chown=uwsgi:uwsgi . .

Creamos la orden de entrada del contenedor, como vemos ejecuta el Shell script que creamos anteriormente..

CMD ["/bin/sh", "/opt/code/docker/start\_server.sh"]

#### instrucciones Docker

Crear un contenedor y hacer que funcione no es una tarea rápida ni sencilla, las siguientes instrucciones nos van a ayudar a depurar el funcionamiento.

**Probar la aplicación**

python3 -m app run

**Compilar:**

DOCKER\_BUILDKIT=1 docker build --tag ms\_csjp\_login .

**Ejecutar para producción**

docker run --rm -d -p 15000:15000 --env-file environment.env --name ms\_csjp\_login\_c ms\_csjp\_login:latest

**Ejecutar para pruebas**

docker run -it -p 15000:15000 --env-file environment.env --name ms\_csjp\_login\_c ms\_csjp\_login:latest

**# El server tiene que escuchar en la ip externa del host, en /etc/mysql/mysqld.conf.d/mysqld.cnf**

bind-address = 127.0.0.1, 192.168.91.130

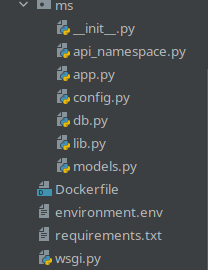
mysqlx-bind-address = 127.0.0.1, 192.168.91.130

**# Las contraseñas de BBDD en formato nativo obligatoriamente**

ALTER USER 'yourusername'@'%' IDENTIFIED WITH mysql\_native\_password BY 'youpassword';

## Servicio rentrada

### Directorio auth

El directorio *ms* contendrá todo el código fuente de nuestro ms. Se va a dividir es siete ficheros:

* **config.py** contendrá los valores de configuración de la aplicación.
* **db.py** define la clase de acceso a las tablas de la base de datos rentrada, así como las configuraciones necesarias para conectarnos al servidor MariaDB o MySQL.
* **models.py** tiene las estructuras de datos necesarias para las comunicaciones, en concreto el modelo de acceso a la base de datos y la estructura de datos del mensaje entre el cliente y el servidor.
* **app.py** es el fichero base de nuestra aplicación, define la función de creación de la aplicación. Esta se basa en el Frameworks flask\_resk.
* **api\_namespace.py** define los puntos de entrada y valores a pasar definidos en el análisis de ms para el usuario.

#### config.py

El fichero de configuración contiene las constantes de configuración tanto del ms que se está creando como los datos del servidor de validación necesario para la comprobación de token.

import os

SRV\_IP = os.environ.get('SRV\_IP', '127.0.0.1')

SRV\_PORT = os.environ.get('SRV\_PORT', '15001')

SRV\_IP\_VALIDATION = os.environ.get('SRV\_IP\_VALIDATION', '127.0.0.1')

SRV\_PORT\_VALIDATION = os.environ.get('SRV\_PORT\_VALIDATION', '15000')

URL\_PATH\_VALIDATION = '/api/v1/validate/'

#### db.py

El fichero de gestión de la BBDD contiene un conjunto de constantes que se establecerán, del mismo modo que en el fichero anterior a través de variables de entorno, para el acceso y la gestión del servidor de la Base de datos (**DB\_XXXX**).

A continuación, crea la configuración (**db\_config**) para el acceso a la base de datos y el objeto que se utilizará para la gestión de la misma dentro de nuestra aplicación (**db**).

Por último, se crea la clase que identifica las tablas *easunto*, *edestino* y *registro* del servidor, es utilizada por la librería SQLAchemy para poder gestionar las órdenes SQL de gestión (INSERT, SELECT, etc) a través del objeto db.

import os

from flask\_sqlalchemy import SQLAlchemy

DB\_USER = os.environ.get('DB\_USER', 'log')

DB\_PW = os.environ.get('DB\_PW', 'log')

DB\_SRV = os.environ.get('DB\_SRV', 'localhost')

DB\_BBDD = os.environ.get('DB\_BBDD', 'rentrada')

DB\_URI = f"mysql://{DB\_USER}:{DB\_PW}@{DB\_SRV}/{DB\_BBDD}"

db\_config = {

'SQLALCHEMY\_DATABASE\_URI': DB\_URI,

'SQLALCHEMY\_TRACK\_MODIFICATIONS': False,

}

db = SQLAlchemy()

def xstr(s):

if s is None:

return ''

else:

return str(s)

class Easunto(db.Model):

id = db.Column(db.Integer(), primary\_key=True)

descri = db.Column(db.String(50), nullable=False)

def \_\_str\_\_(self):

return "{\"id\":" + str(self.id) + ", \"descri\":\"" + self.descri + "\"}"

class Edestino(db.Model):

id = db.Column(db.Integer(), primary\_key=True)

descri = db.Column(db.String(50), nullable=False)

def \_\_str\_\_(self):

return "{\"id\":" + str(self.id) + ", \"descri\":\"" + self.descri + "\"}"

class Registro(db.Model):

referencia = db.Column(db.Integer(), primary\_key=True)

fecha = db.Column(db.String(25), nullable=False)

asunto = db.Column(db.Integer(), nullable=False)

asuntootro = db.Column(db.String(50))

destino = db.Column(db.Integer(), nullable=False)

destinootro = db.Column(db.String(50), nullable=False)

interesado = db.Column(db.String(50), nullable=False)

resumen = db.Column(db.String(100))

observaciones = db.Column(db.Text)

fecham = db.Column(db.String(25))

mmod = db.Column(db.Integer())

def \_\_str\_\_(self):

return "{\"referencia\":\"" + xstr(self.referencia) + \

"\", \"fecha\":\"" + xstr(self.fecha) + \

"\", \"asunto\":\"" + xstr(self.asunto) + \

"\", \"asuntootro\":\"" + xstr(self.asuntootro) + \

"\", \"destino\":\"" + xstr(self.destino) + \

"\", \"destinootro\":\"" + xstr(self.destinootro) + \

"\", \"interesado\":\"" + xstr(self.interesado) + \

"\", \"resumen\":\"" + xstr(self.resumen) + \

"\", \"observaciones\":\"" + xstr(self.observaciones) + \

"\", \"fecham\":\"" + xstr(self.fecham) + \

"\", \"mmod\":\"" + xstr(self.mmod) + \

"\"}"

#### models.py

El fichero models va a implementar las estructuras de datos de gestión de nuestra aplicación. De este fichero son relevantes los payload. El **return\_payload** contiene la estructura de campos que se va a utilizar en la respuesta al cliente en cualquiera de las peticiones al servidor.

from flask\_restx import fields

return\_payload = {

'message': fields.String()

}

#### app.py

El fichero principal de creación de nuestra aplicación. Simplemente exporta una función para crear las estructuras flask de la misma. Esta función es llamada en el momento de la primera ejecución de nuestro servidor uwsgi. Cuando un usuario acceda a una url del servidor se buscará a través de los puntos de acceso registrados en los ficheros api\_namespace y si hay coincidencia se ejecutará la función asignada.

#### Lib.py

El fichero de librería, tiene un conjunto de funciones de apoyo a la aplicación.

from http import HTTPStatus

import requests

from ms.db import db, Easunto, Edestino, Registro

from sqlalchemy.sql.expression import func

from ms.config import SRV\_IP\_VALIDATION, SRV\_PORT\_VALIDATION, URL\_PATH\_VALIDATION

import json

La siguiente función validará contra el ms de validación el token recibido por parte del usuario para realizar la acción solicitada

def validate\_token(token):

new\_data = {

'token': token,

}

url = f'http://{SRV\_IP\_VALIDATION}:{SRV\_PORT\_VALIDATION}{URL\_PATH\_VALIDATION}'

try:

respuesta = requests.get(url, params=new\_data)

return respuesta.status\_code

except:

return HTTPStatus.FORBIDDEN

Devuelve todas las filas de la tabla asuntos para usar por parte del cliente.

def get\_asuntos():

asuntos = Easunto.query.all()

data = ""

for asunto in asuntos:

data += str(asunto) + ','

return "[" + data[:-1] + "]" # eliminando la coma final de más

Devuelve todas las filas de la tabla destinos para usar por parte del cliente.

def get\_destinos():

destinos = Edestino.query.all()

data = ""

for destino in destinos:

data += str(destino) + ','

return "[" + data[:-1] + "]" # eliminando la coma final de más

Devuelve todas las líneas del registro de entrada.

def get\_rows():

rows = Registro.query.all()

data = ""

for row in rows:

data += str(row) + ','

return "[" + data[:-1] + "]" # eliminando la coma final de más

Devuelve todos los datos de una de las filas del registro de entrada.

def get\_row(identificador):

row = Registro.query.filter\_by(referencia=identificador).first()

data = ""

if row:

data = str(row)

return HTTPStatus.OK, "[" + data + "]"

return HTTPStatus.NOT\_FOUND, data

Inserta en el registro de entrada una línea.

def insert\_row(data):

referencia = 0

if db.session.query(func.max(Registro.referencia)).scalar() is not None:

referencia = db.session.query(func.max(Registro.referencia)).scalar() + 1

new\_row = Registro(

referencia=referencia,

fecha=data['fecha'].strftime('%Y-%m-%d'),

asunto=data['asunto'],

asuntootro=data['asuntootro'],

destino=data['destino'],

destinootro=data['destinootro'],

interesado=data['interesado'],

resumen=data['resumen'],

observaciones=data['observaciones'],

fecham=data['fecham'] .strftime('%Y-%m-%d'),

mmod=data['mmod'],

)

try:

db.session.add(new\_row)

db.session.commit()

except:

return HTTPStatus.FORBIDDEN

return HTTPStatus.OK

Actualiza los datos de una fila del registro de entrada.

def update\_row(identificador, data):

row = Registro.query.filter\_by(referencia=identificador).first()

if row:

row.fecha=data['fecha'].strftime('%Y-%m-%d')

row.asunto=data['asunto']

row.asuntootro=data['asuntootro']

row.destino=data['destino']

row.destinootro=data['destinootro']

row.interesado=data['interesado']

row.resumen=data['resumen']

row.observaciones=data['observaciones']

row.fecham=data['fecham'] .strftime('%Y-%m-%d')

row.mmod=data['mmod']

try:

db.session.commit()

except:

return HTTPStatus.FORBIDDEN

return HTTPStatus.OK

return HTTPStatus.FORBIDDEN

#### api\_namespace.py

Registra las url de acceso por el servidor web del usuario.

from datetime import datetime

from http import HTTPStatus

from flask\_restx import Namespace, Resource, abort

from flask import request, make\_response

from ms.models import return\_payload

from ms.lib import validate\_token, get\_asuntos, \

get\_destinos, get\_rows, get\_row, insert\_row, update\_row

from flask\_restx import reqparse

api\_namespace = Namespace('api', description='User API operations')

return\_payload\_model = api\_namespace.model('Return', return\_payload)

Url de acceso a los asuntos del registro de entrada, todas las llamadas primero validan el token, en caso correcto se ejecutará la acción solicitada.

@api\_namespace.route('/v1/get\_asuntos/')

class GetAsuntosMs(Resource):

@api\_namespace.doc('get\_asuntos')

@api\_namespace.doc(params={'token': 'Token de usuario'})

@api\_namespace.response(200, 'Asuntos', return\_payload\_model)

@api\_namespace.response(403, 'Error de acceso', return\_payload\_model)

@api\_namespace.marshal\_with(return\_payload\_model)

def get(self):

token = request.args.get("token", None)

code = validate\_token(token)

if code == HTTPStatus.OK:

return\_payload['message'] = get\_asuntos()

return return\_payload

abort(code)

@api\_namespace.route('/v1/get\_destinos/')

class GetDestinosMs(Resource):

@api\_namespace.doc('get\_destinos')

@api\_namespace.doc(params={'token': 'Token de usuario'})

@api\_namespace.response(200, 'Destinos', return\_payload\_model)

@api\_namespace.response(403, 'Error de acceso', return\_payload\_model)

@api\_namespace.marshal\_with(return\_payload\_model)

def get(self):

token = request.args.get("token", None)

code = validate\_token(token)

if code == HTTPStatus.OK:

return\_payload['message'] = get\_destinos()

return return\_payload

abort(code)

@api\_namespace.route('/v1/get\_rows/')

class GetRowsMs(Resource):

@api\_namespace.doc('get\_rows')

@api\_namespace.doc(params={'token': 'Token de usuario'})

@api\_namespace.response(200, 'Get Rows', return\_payload\_model)

@api\_namespace.response(403, 'Error de acceso', return\_payload\_model)

@api\_namespace.marshal\_with(return\_payload\_model)

def get(self):

token = request.args.get("token", None)

code = validate\_token(token)

if code == HTTPStatus.OK:

return\_payload['message'] = get\_rows()

return return\_payload

abort(code)

@api\_namespace.route('/v1/get\_rows/<int:id>/')

class GetRowsByIdMs(Resource):

@api\_namespace.doc('get\_rows\_by\_id')

@api\_namespace.doc(params={'token': 'Token de usuario'})

@api\_namespace.response(200, 'Get Rows by id', return\_payload\_model)

@api\_namespace.response(403, 'Error de acceso', return\_payload\_model)

@api\_namespace.marshal\_with(return\_payload\_model)

def get(self, id):

token = request.args.get("token", None)

code = validate\_token(token)

if code == HTTPStatus.OK:

code\_row, row = get\_row(id)

if code\_row == HTTPStatus.OK:

return\_payload['message'] = row

return return\_payload

abort(code\_row)

abort(code)

@api\_namespace.route('/v1/insert\_row/')

class InsertRowMs(Resource):

@api\_namespace.doc('insert\_rows')

@api\_namespace.doc(params={'token': 'Token de usuario',

'fecha': 'fecha',

'asunto': 'asunto',

'asuntootro': 'asuntootro',

'destino': 'destino',

'destinootro': 'destinootro',

'interesado': 'interesado',

'resumen': 'resumen',

'observaciones': 'observaciones',

'fecham': 'fecham',

'mmod': 'mmod'

})

@api\_namespace.response(200, 'Insert rows', return\_payload\_model)

@api\_namespace.response(403, 'Error de acceso', return\_payload\_model)

@api\_namespace.marshal\_with(return\_payload\_model)

def get(self):

token = request.args.get("token", None)

code = validate\_token(token)

if code == HTTPStatus.OK:

parser = reqparse.RequestParser()

parser.add\_argument('fecha',

type=lambda x: datetime.strptime(x, '%Y-%m-%d'), required=True, help='Fecha')

parser.add\_argument('asunto', type=int, required=True, help='asunto')

parser.add\_argument('asuntootro', type=str, help='asuntootro')

parser.add\_argument('destino', type=int, required=True, help='destino')

parser.add\_argument('destinootro', type=str, help='destinootro')

parser.add\_argument('interesado', type=str, required=True, help='interesado')

parser.add\_argument('resumen', type=str, help='resumen')

parser.add\_argument('observaciones', type=str, help='observaciones')

parser.add\_argument('fecham',

type=lambda x: datetime.strptime(x, '%Y-%m-%d'), help='fecham')

parser.add\_argument('mmod', type=str, help='mmod')

args = parser.parse\_args()

code\_row = insert\_row(args)

if code\_row == HTTPStatus.OK:

return\_payload['message'] = HTTPStatus.OK

return return\_payload

abort(code\_row)

abort(code)

@api\_namespace.route('/v1/update\_row/<int:id>/')

class InsertRowMs(Resource):

@api\_namespace.doc('update\_rows')

@api\_namespace.doc(params={'token': 'Token de usuario',

'id': 'id',

'fecha': 'fecha',

'asunto': 'asunto',

'asuntootro': 'asuntootro',

'destino': 'destino',

'destinootro': 'destinootro',

'interesado': 'interesado',

'resumen': 'resumen',

'observaciones': 'observaciones',

'fecham': 'fecham',

'mmod': 'mmod'

})

@api\_namespace.response(200, 'Update rows', return\_payload\_model)

@api\_namespace.response(403, 'Error de acceso', return\_payload\_model)

@api\_namespace.marshal\_with(return\_payload\_model)

def get(self, id):

token = request.args.get("token", None)

code = validate\_token(token)

if code == HTTPStatus.OK:

parser = reqparse.RequestParser()

parser.add\_argument('fecha',

type=lambda x: datetime.strptime(x, '%Y-%m-%d'), required=True, help='Fecha')

parser.add\_argument('asunto', type=int, required=True, help='asunto')

parser.add\_argument('asuntootro', type=str, help='asuntootro')

parser.add\_argument('destino', type=int, required=True, help='destino')

parser.add\_argument('destinootro', type=str, help='destinootro')

parser.add\_argument('interesado', type=str, required=True, help='interesado')

parser.add\_argument('resumen', type=str, help='resumen')

parser.add\_argument('observaciones', type=str, help='observaciones')

parser.add\_argument('fecham',

type=lambda x: datetime.strptime(x, '%Y-%m-%d'), help='fecham')

parser.add\_argument('mmod', type=str, help='mmod')

args = parser.parse\_args()

code\_row = update\_row(id, args)

if code\_row == HTTPStatus.OK:

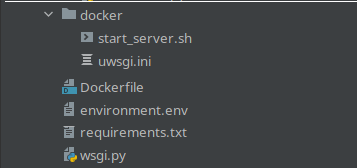
return\_payload['message'] = HTTPStatus.OK

return return\_payload

abort(code\_row)

abort(code)

### Dockerizar el servicio

Hay pocas diferencias entre dockerizar este ms y el creado anteriormente, añadir las variables de entorno necesarias y cambiar los parámetros.

#### environment.env

Este fichero contendrá todas las variables de entorno que queremos crear en el momento de la ejecución de nuestro contenedor, son las mismas que ya se definieron dentro del ms. De esta manera permitimos configuraciones externas a nuestro ms de forma automática.

SRV\_IP=127.0.0.1

SRV\_PORT=15001

SRV\_IP\_VALIDATION=192.168.91.130

SRV\_PORT\_VALIDATION=15000

DB\_USER=log

DB\_PW=log

DB\_SRV=192.168.91.130

DB\_BBDD=rentrada

#### instrucciones Docker

Crear un contenedor y hacer que funcione no es una tarea rápida ni sencilla, las siguientes instrucciones nos van a ayudar a depurar el funcionamiento.

**Probar la aplicación**

python3 -m app run

**Compilar:**

DOCKER\_BUILDKIT=1 docker build --tag ms\_csjp\_rentrada .

**Ejecutar para producción**

docker run --rm -d -p 15001:15001 --env-file environment.env --name ms\_csjp\_rentrada\_c ms\_csjp\_rentrada:latest

**Ejecutar para pruebas**

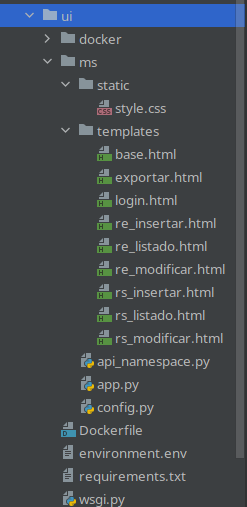
docker run -it -p 15001:15001 --env-file environment.env --name ms\_csjp\_rentrada\_c ms\_csjp\_rentrada:latest

## Servicio rsalida

Para ver la implementación de este ms hay que dirigirse al código fuente, los cambios son mínimos con respecto al ms rentrada, el puerto, las clases de gestión de la base de datos y poco más.

## Servicio ui usuario

### Directorio ui

El servicio ui contendrá el cliente que utilizaremos de ejemplo para la aplicación, en este caso es un cliente html/javascript generado a través del framework Flask que se ejecutará dentro de un contenedor al igual que los demás a modo de ejemplo, pero podría ser cualquier aplicación que usase los ms rentrada/rsalida/login directamente:

* **/static** simplemente tendrá el fichero css de estilo.
* **/templates** contendrá todas las plantillas de la aplicación en formato jinja2 para usar con flask. Las plantillas se entienden por nombre, solo hay que tener en cuenta que *re* hace referencia a registro de entrada y *rs* a registro de salida.
* **config.py** tiene las variables de configuración del ms.
* **app.py** es el fichero base de nuestra aplicación, define la función de creación de la aplicación. Esta se basa en el Frameworks flask\_resk.
* **api\_namespace.py** define los puntos de entrada y valores a pasar definidos en el análisis de ms para el usuario.

#### config.py

El fichero de configuración contiene las constantes de configuración tanto del ms que se está creando como los datos de los ms necesarios para el funcionamiento.

import os

SRV\_IP = os.environ.get('SRV\_IP', '127.0.0.1')

SRV\_PORT = os.environ.get('SRV\_PORT', '8080')

SRV\_IP\_VALIDATION = os.environ.get('SRV\_IP\_VALIDATION', '127.0.0.1')

SRV\_PORT\_VALIDATION = os.environ.get('SRV\_PORT\_VALIDATION', '15000')

SRV\_IP\_RENTRADA = os.environ.get('SRV\_IP\_RENTRADA', '127.0.0.1')

SRV\_PORT\_RENTRADA = os.environ.get('SRV\_PORT\_RENTRADA', '15001')

SRV\_IP\_RSALIDA = os.environ.get('SRV\_IP\_RSALIDA', '127.0.0.1')

SRV\_PORT\_RSALIDA = os.environ.get('SRV\_PORT\_RSALIDA', '15002')

URL\_PATH\_VALIDATION = '/api/v1/validate/'

URL\_PATH\_LOGIN = '/api/v1/login/'

URL\_PATH\_GET\_ROWS = '/api/v1/get\_rows/'

URL\_PATH\_INSERT\_ROWS = '/api/v1/insert\_row/'

URL\_PATH\_UPDATE\_ROWS = '/api/v1/update\_row/'

URL\_PATH\_GET\_ASUNTOS = '/api/v1/get\_asuntos/'

URL\_PATH\_GET\_DESTINOS = '/api/v1/get\_destinos/'

#### api\_namespace.py

Registra las url de acceso por el servidor web del usuario.

import functools

import os

from http import HTTPStatus

from ms.config import \

SRV\_IP\_VALIDATION, SRV\_PORT\_VALIDATION, \

SRV\_IP\_RENTRADA, SRV\_PORT\_RENTRADA, \

SRV\_IP\_RSALIDA, SRV\_PORT\_RSALIDA, \

URL\_PATH\_LOGIN,URL\_PATH\_GET\_ROWS, \

URL\_PATH\_INSERT\_ROWS, URL\_PATH\_UPDATE\_ROWS, \

URL\_PATH\_GET\_ASUNTOS, URL\_PATH\_GET\_DESTINOS

import requests

from flask import Blueprint, flash, g, redirect, render\_template, \

request, session, url\_for, make\_response

import json

import io

import csv

import logging

logging.basicConfig(level=os.environ.get("LOGLEVEL", "INFO"))

logger = logging.getLogger(\_\_name\_\_)

bp = Blueprint('ui', \_\_name\_\_, url\_prefix='/')

def login\_required(view):

@functools.wraps(view)

def wrapped\_view(\*\*kwargs):

if g.token is None:

return redirect(url\_for('ui.login'))

return view(\*\*kwargs)

return wrapped\_view

def get\_table\_aux(rentrada=True, asuntos=True):

"""

Devuelve el contenido de las tablas asuntos y destinos de los registros de entrada y salida

"""

new\_data = {

'token': g.token,

}

if rentrada:

url = f'http://{SRV\_IP\_RENTRADA}:{SRV\_PORT\_RENTRADA}'

else:

url = f'http://{SRV\_IP\_RSALIDA}:{SRV\_PORT\_RSALIDA}'

if asuntos:

url += f"{URL\_PATH\_GET\_ASUNTOS}"

else:

url += f"{URL\_PATH\_GET\_DESTINOS}"

try:

logger.info('get\_table\_aux:'+url)

respuesta = requests.get(url, params=new\_data)

except Exception as e:

logger.info('get\_table\_aux:' + str(e))

return []

logger.info('get\_table\_aux:' + str(respuesta.content))

if respuesta.status\_code == HTTPStatus.OK:

return json.loads(json.loads(respuesta.content)['message'])

return []

@bp.before\_app\_request # Se ejecuta siempre antes de cualquier enrutado

def load\_logged\_in\_user():

token = session.get('token')

if token is None:

g.token = None

else:

g.token = token

@bp.route('/')

def index():

return render\_template('base.html')

@bp.route('/login', methods=('GET', 'POST'))

def login():

if request.method == 'POST':

username = request.form['username']

password = request.form['password']

error = None

if not username:

error = 'Username is required.'

elif not password:

error = 'Password is required.'

if error is None:

session.clear()

# Intentamos validar contra el servidor de validación

new\_data = {

'user': username,

'pass': password,

}

url = f'http://{SRV\_IP\_VALIDATION}:{SRV\_PORT\_VALIDATION}{URL\_PATH\_LOGIN}'

try:

logger.info('login:' + url)

respuesta = requests.get(url, params=new\_data)

except Exception as e:

return redirect(url\_for("ui.login"))

if respuesta.status\_code == HTTPStatus.OK:

session['token'] = json.loads(respuesta.content)['message']

return redirect(url\_for("ui.index"))

flash(error)

return render\_template("login.html")

@bp.route('/logout')

@login\_required

def logout():

session.clear()

return redirect(url\_for('ui.index'))

@bp.route('/exportar', methods=('GET', 'POST'))

@login\_required

def exportar():

if request.method == 'POST':

new\_data = {

'token': g.token,

}

url = f'http://{SRV\_IP\_RENTRADA}:{SRV\_PORT\_RENTRADA}{URL\_PATH\_GET\_ROWS}'

try:

logger.info('exportar\_1:' + url)

respuesta = requests.get(url, params=new\_data)

except Exception as e:

logger.info('exportar\_2:' + str(e))

return redirect(url\_for("ui.index"))

logger.info('exportar\_3:' + str(respuesta.content))

if respuesta.status\_code == HTTPStatus.OK:

# La respuesta es json, pero el mensaje también lo es, por eso dos conversiones

data = json.loads(json.loads(respuesta.content)['message'])

si = io.StringIO()

cw = csv.writer(si)

imprimir\_cabecera = True

for linea in data:

if imprimir\_cabecera:

cw.writerow(linea.keys())

imprimir\_cabecera = False

cw.writerow(linea.values())

output = make\_response(si.getvalue())

output.headers["Content-Disposition"] = "attachment; filename=export.csv"

output.headers["Content-type"] = "text/csv"

return output

return render\_template("exportar.html")

@bp.route('/rentrada/listado')

@login\_required

def rentrada\_listado():

new\_data = {

'token': g.token,

}

url = f'http://{SRV\_IP\_RENTRADA}:{SRV\_PORT\_RENTRADA}{URL\_PATH\_GET\_ROWS}'

try:

logger.info('rentrada\_listado:' + url)

respuesta = requests.get(url, params=new\_data)

except Exception as e:

return redirect(url\_for("ui.index"))

if respuesta.status\_code == HTTPStatus.OK:

# La respuesta es json, pero el mensaje también lo es, por eso dos conversiones

data = json.loads(json.loads(respuesta.content)['message'])

return render\_template("re\_listado.html", data=data)

return render\_template("re\_listado.html")

@bp.route('/rentrada/insertar', methods=('GET', 'POST'))

@login\_required

def rentrada\_insertar():

if request.method == 'POST':

new\_data = {

'token': g.token,

'fecha': request.form['fecha'],

'asunto': request.form['asunto'],

'asuntootro': request.form['asuntootro'],

'destino': request.form['destino'],

'destinootro': request.form['destinootro'],

'interesado': request.form['interesado'],

'resumen': request.form['resumen'],

'observaciones': request.form['observaciones'],

'mmod': request.form['mmod']

}

if len(request.form['fecham']):

new\_data['fecham'] = request.form['fecham']

url = f'http://{SRV\_IP\_RENTRADA}:{SRV\_PORT\_RENTRADA}{URL\_PATH\_INSERT\_ROWS}'

try:

logger.info('rentrada\_insertar:' + url)

respuesta = requests.get(url, params=new\_data)

except Exception as e:

return redirect(url\_for("ui.index"))

if respuesta.status\_code == HTTPStatus.OK:

return redirect(url\_for("ui.rentrada\_listado"))

return render\_template("re\_insertar.html",

asuntos=get\_table\_aux(), destinos=get\_table\_aux(asuntos=False))

@bp.route('/rentrada/modificar/<int:id>/', methods=('GET', 'POST'))

@login\_required

def rentrada\_modificar(id):

data = None

if request.method == 'POST':

new\_data = {

'token': g.token,

'id': id,

'fecha': request.form['fecha'],

'asunto': request.form['asunto'],

'asuntootro': request.form['asuntootro'],

'destino': request.form['destino'],

'destinootro': request.form['destinootro'],

'interesado': request.form['interesado'],

'resumen': request.form['resumen'],

'observaciones': request.form['observaciones'],

'fecham': request.form['fecham'],

'mmod': request.form['mmod']

}

url = \

f'http://{SRV\_IP\_RENTRADA}:{SRV\_PORT\_RENTRADA}{URL\_PATH\_UPDATE\_ROWS}{str(id)}/'

try:

logger.info('rentrada\_modificar\_post:' + url)

respuesta = requests.get(url, params=new\_data)

except Exception as e:

return redirect(url\_for("ui.index"))

if respuesta.status\_code == HTTPStatus.OK:

return redirect(url\_for("ui.rentrada\_listado"))

new\_data = {

'token': g.token,

}

url = f'http://{SRV\_IP\_RENTRADA}:{SRV\_PORT\_RENTRADA}{URL\_PATH\_GET\_ROWS}{str(id)}/'

try:

logger.info('rentrada\_modificar\_get:' + url)

respuesta = requests.get(url, params=new\_data)

except Exception as e:

return redirect(url\_for("ui.index"))

if respuesta.status\_code == HTTPStatus.OK:

data = json.loads(json.loads(respuesta.content)['message'])[0]

return render\_template("re\_modificar.html", data=data,

asuntos=get\_table\_aux(), destinos=get\_table\_aux(asuntos=False))

@bp.route('/rsalida/listado')

@login\_required

def rsalida\_listado():

new\_data = {

'token': g.token,

}

url = f'http://{SRV\_IP\_RSALIDA}:{SRV\_PORT\_RSALIDA}{URL\_PATH\_GET\_ROWS}'

try:

respuesta = requests.get(url, params=new\_data)

except Exception as e:

return redirect(url\_for("ui.index"))

if respuesta.status\_code == HTTPStatus.OK:

# La respuesta es json, pero el mensaje también lo es, por eso dos conversiones

data = json.loads(json.loads(respuesta.content)['message'])

return render\_template("rs\_listado.html", data=data)

return render\_template("rs\_listado.html")

@bp.route('/rsalida/insertar', methods=('GET', 'POST'))

@login\_required

def rsalida\_insertar():

if request.method == 'POST':

new\_data = {

'token': g.token,

'fecha': request.form['fecha'],

'destino': request.form['destino'],

'destinootro': request.form['destinootro'],

'interesado': request.form['interesado'],

'observaciones': request.form['observaciones'],

'mmod': request.form['mmod']

}

if len(request.form['fecham']):

new\_data['fecham'] = request.form['fecham']

url = f'http://{SRV\_IP\_RSALIDA}:{SRV\_PORT\_RSALIDA}{URL\_PATH\_INSERT\_ROWS}'

try:

respuesta = requests.get(url, params=new\_data)

except Exception as e:

return redirect(url\_for("ui.index"))

if respuesta.status\_code == HTTPStatus.OK:

return redirect(url\_for("ui.rsalida\_listado"))

return render\_template("rs\_insertar.html",

destinos=get\_table\_aux(rentrada=False, asuntos=False))

@bp.route('/rsalida/modificar/<int:id>/', methods=('GET', 'POST'))

@login\_required

def rsalida\_modificar(id):

data = None

if request.method == 'POST':

new\_data = {

'token': g.token,

'id': id,

'fecha': request.form['fecha'],

'destino': request.form['destino'],

'destinootro': request.form['destinootro'],

'interesado': request.form['interesado'],

'observaciones': request.form['observaciones'],

'fecham': request.form['fecham'],

'mmod': request.form['mmod']

}

url = f'http://{SRV\_IP\_RSALIDA}:{SRV\_PORT\_RSALIDA}{URL\_PATH\_UPDATE\_ROWS}{str(id)}/'

try:

respuesta = requests.get(url, params=new\_data)

except Exception as e:

return redirect(url\_for("ui.index"))

if respuesta.status\_code == HTTPStatus.OK:

return redirect(url\_for("ui.rsalida\_listado"))

new\_data = {

'token': g.token,

}

url = f'http://{SRV\_IP\_RSALIDA}:{SRV\_PORT\_RSALIDA}{URL\_PATH\_GET\_ROWS}{str(id)}/'

try:

respuesta = requests.get(url, params=new\_data)

except Exception as e:

return redirect(url\_for("ui.index"))

if respuesta.status\_code == HTTPStatus.OK:

data = json.loads(json.loads(respuesta.content)['message'])[0]

return render\_template("rs\_modificar.html", data=data,

destinos=get\_table\_aux(rentrada=False, asuntos=False))

#### instrucciones Docker

**Probar la aplicación**

python3 -m app run

**Compilar:**

DOCKER\_BUILDKIT=1 docker build --tag ms\_csjp\_ui .

**Ejecutar para producción**

docker run --rm -d -p 8080:8080 --env-file environment.env --name ms\_csjp\_ui\_c ms\_csjp\_rentrada:latest

**Ejecutar para pruebas**

docker run -it -p 8080:8080 --env-file environment.env --name ms\_csjp\_ui\_c ms\_csjp\_ui:latest

## Servicio ui admin

Se deja al lector el crear este servicio bajo el puerto 8081.

## Ejecución con docker-compose

### docker-compose.yaml

version: '3.8'

services:

ms\_csjp\_login\_c:

env\_file: ./autorizacion/environment.env

image: ms\_csjp\_login

build:

context: ./autorizacion

dockerfile: Dockerfile

ports:

- "15000:15000"

restart: always

ms\_csjp\_rentrada\_c:

env\_file: ./rentrada/environment.env

image: ms\_csjp\_rentrada

build:

context: ./rentrada

dockerfile: Dockerfile

ports:

- "15001:15001"

depends\_on:

- "ms\_csjp\_login\_c"

restart: always

ms\_csjp\_rsalida\_c:

env\_file: ./rsalida/environment.env

image: ms\_csjp\_rsalida

build:

context: ./rsalida

dockerfile: Dockerfile

ports:

- "15002:15002"

depends\_on:

- "ms\_csjp\_login\_c"

restart: always

ms\_csjp\_ui\_c:

env\_file: ./ui/environment.env

image: ms\_csjp\_ui

build:

context: ./ui

dockerfile: Dockerfile

ports:

- "8080:8080"

depends\_on:

- "ms\_csjp\_login\_c"

- "ms\_csjp\_rentrada\_c"

- "ms\_csjp\_rsalida\_c"

restart: always

**Ejecutar la aplicación**

$**docker-compose up -d**

docker-compose up -d

Starting pythonproject\_ms\_csjp\_login\_c\_1 ... done

Starting pythonproject\_ms\_csjp\_rentrada\_c\_1 ... done

Starting pythonproject\_ms\_csjp\_rsalida\_c\_1 ... done

Starting pythonproject\_ms\_csjp\_ui\_c\_1 ... done

**Parar la aplicación**

**$docker-compose down**

Stopping pythonproject\_ms\_csjp\_ui\_c\_1 ... done

Stopping pythonproject\_ms\_csjp\_rsalida\_c\_1 ... done

Stopping pythonproject\_ms\_csjp\_rentrada\_c\_1 ... done

Stopping pythonproject\_ms\_csjp\_login\_c\_1 ... done

Removing pythonproject\_ms\_csjp\_ui\_c\_1 ... done

Removing pythonproject\_ms\_csjp\_rsalida\_c\_1 ... done

Removing pythonproject\_ms\_csjp\_rentrada\_c\_1 ... done

Removing pythonproject\_ms\_csjp\_login\_c\_1 ... done

Removing network pythonproject\_default