



UNIVERSIDAD DE BURGOS  
ESCUELA POLITÉCNICA SUPERIOR  
Grado en Ingeniería en Informática



TFG del Grado en Ingeniería Informática

Framework web de uso de  
sistemas de machine learning.



Presentado por Javier Martínez Riberas  
en Universidad de Burgos — 21 de junio de 2017  
Tutor: Dr. José Francisco Díez Pastor y Dr. César  
Ignacio García Osorio

Copyright (C) 2017 Javier Martínez Riberas <[javyermartinez@gmail.com](mailto:javyermartinez@gmail.com)>. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".





UNIVERSIDAD DE BURGOS  
ESCUELA POLITÉCNICA SUPERIOR  
Grado en Ingeniería en Informática



D. José Francisco Díez Pastor y D. César Ignacio García Osorio, profesores del departamento de Ingeniería Civil, área de Lenguajes y Sistemas Informáticos.

Expone:

Que el alumno D. Javier Martínez Riberas, con DNI 71299495R, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado ” .

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 21 de junio de 2017

Vº. Bº. del Tutor:

Vº. Bº. del co-tutor:

D. José Francisco Díez Pastor

D. César Ignacio García Osorio



## Resumen

Recientemente la mayoría de empresas que usan productos informáticos están intentando mejorar sus servicios con un enfoque ‘AI first’, esto es más prevalente en los grandes del sector que tratan de aplicar el aprendizaje automático de manera más amplia.

La escalabilidad es un concepto con una definición clara pero muy abstracta [7], la mejor definición que se ha encontrado se atribuye a Bondi [2] y la define como la capacidad de un sistema para ser capaz de ampliarse para manejar mayor cantidad de trabajo.

Desde el punto de vista de mantenimiento un sistema escalable es aquel al que podemos mantener o incrementar su funcionalidad sin incurrir en una cantidad de deuda técnica demasiado grande permitiendo un desarrollo más rápido.

La escalabilidad es una propiedad compleja en el mundo software ya que existen varias fuentes de ampliación del sistema, la más común es la escalabilidad aumentando el número de copias del software que se ejecutan, en el mismo ordenador o en distintos ordenadores. Se puede descomponer el sistema en los subsistemas más pequeños posibles de manera que la complejidad disminuye convirtiendo el sistema más fácil de mantener. También existe una última que es separar los servicios en particiones distintas de manera que cada partición se encarga de dar servicio a parte de los datos.

Estas tres opciones de escalado se conocen en [1] como el cubo de la escalabilidad.

En este trabajo se busca obtener un sistema escalable en los dos primeros ejes de escalado. El primer eje o dimensión lo vamos a conseguir con contenedores, el sistema que usaremos es docker. La segunda dimensión la obtendremos con la arquitectura de microservicios.

Esto busca dar un fácil acceso a un problema común en el campo del aprendizaje automático que es la clasificación de imágenes. Esto se hace mediante Deep Learning con el framework Tensorflow.

Con esto se pretende dar un ejemplo de una de las muchas maneras de evitar problemas de escalabilidad, un problema bastante común en el mundo del desarrollo software. Esta escalabilidad es tanto de rendimiento como de mantenimiento. Se busca facilitar una arquitectura fácilmente escalable en ambos sentidos cuyo reflejo en código no es complejo para el futuro aprendizaje. También se intenta proporcionar un esqueleto mutable para que los futuros proyectos que quieran usar un control básico de usuarios puedan usar para centrarse en otras partes o temas de mayor interés.

Para exponer cómo se resuelve el añadir más servicios se ha expuesto un servicio que usa Deep Learning para clasificar imágenes sobre un conjunto de clases (Imagenet [8]).

## **Descriptores**

Minería de datos, redes neuronales, clasificadores, aplicación web. . .



**Abstract**

A **brief** presentation of the topic addressed in the project.

**Keywords**

keywords separated by commas.

---

# Índice general

---

Índice general	v
Índice de figuras	vii
Índice de tablas	viii
Introducción	1
Objetivos del proyecto	3
2.1. Objetivos principales	3
2.2. Servidor web y página web	3
2.3. Servicio de Minería	4
2.4. Proceso de desarrollo	4
2.5. Objetivos personales	4
Conceptos teóricos	5
3.1. Deuda técnica	5
3.2. Integración continua ( <i>Continuous Integration</i> )	6
3.3. DevOps	7
3.4. Microservicios	8
Técnicas y herramientas	11
4.1. Metodología	11
4.2. Patrones de diseño	12
4.3. Control de versiones	12
4.4. Integración continua	13
4.5. Quality Assurance	13
4.6. Tests	13
4.7. Dependencias	14
4.8. Comunicación	14

4.9. Documentación . . . . .	14
4.10. Editor de texto . . . . .	14
4.11. Bibliotecas . . . . .	15
<b>Aspectos relevantes del desarrollo del proyecto</b>	<b>16</b>
5.1. Idea e inicio . . . . .	16
5.2. Formación . . . . .	16
5.3. Página web y versionado de bases de datos . . . . .	17
5.4. Heroku y transición a contenedores . . . . .	18
5.5. Docker compose y microservicios . . . . .	19
5.6. Orquestración . . . . .	19
5.7. Conclusiones . . . . .	20
<b>GNU Free Documentation License</b>	<b>22</b>
1. APPLICABILITY AND DEFINITIONS . . . . .	23
2. VERBATIM COPYING . . . . .	24
3. COPYING IN QUANTITY . . . . .	25
4. MODIFICATIONS . . . . .	25
5. COMBINING DOCUMENTS . . . . .	28
6. COLLECTIONS OF DOCUMENTS . . . . .	28
7. AGGREGATION WITH INDEPENDENT WORKS . . . . .	28
8. TRANSLATION . . . . .	29
9. TERMINATION . . . . .	29
10. FUTURE REVISIONS OF THIS LICENSE . . . . .	30
11. RELICENSING . . . . .	30
ADDENDUM: How to use this License for your documents . . . . .	31
<b>Bibliografía</b>	<b>32</b>

---

## Índice de figuras

---

3.1. Deuda técnica: Beneficiosa y perjudicial. [11] . . . . .	6
3.2. Deuda técnica: Techo y base [11] . . . . .	7
3.3. Arquitecturas monolítica y microservicios. . . . .	9

---

# Índice de tablas

---

---

# Introducción

---

El uso mayoritario del aprendizaje automático o machine learning (ML) se puede ver en empresas como Google que ha pasado de buscar el termino exacto pedido por el usuario a intentar, con la información de que disponen de averiguar en que contexto se esta buscando por ejemplo si predice que eres programador y buscas R probablemente los primeros resultados sean del lenguaje de programación.

El servicio a dar acceso es un sistema de machine learning y no de otra clase, por la actualidad y relevancia de la tecnología. El sistema de aprendizaje automático será un clasificador que agrupará las imagenes en los conjuntos de Imagenet [8].

La escalabilidad es muy importante en el desarrollo software actualmente debido a la necesidad de hacer sistemas cada vez más grandes y complejos que a la vez no gasten recursos de manera excesiva. Esto se puede conseguir con ciertas arquitecturas como microservicios. Otras arquitecturas que proporcionan este tipo de ventajas son Serverless [6] [9].

Este trabajo se orienta a conseguir un sistema mediante métodos actuales y escalable. Se intentará que los métodos para conseguir este sistema sean lo más actuales, prestigiosos y usados posibles, ya que probablemente esos métodos serán fundamentales el día de mañana. También se persigue adquirir los conocimientos necesarios para poder replicar este sistema sobre proyectos ya creados.

El sistema en cuestión será una página web, ya que es un sistema altamente accesible (desde casi cualquier plataforma) con mayor facilidad de mantenimiento que la mayor parte de aplicaciones especificas a un dispositivo concreto. Otra ventaja que proporcionan las páginas web es que tendremos acceso a todos los errores y fallos que surjan en ejecución.

Se usarán tecnologías puntas para conseguir estos objetivos. Como docker para la escalabilidad y reducción de deuda técnica gracias a la arquitectura

de microservicios que facilita.

La metodología que se usará será la integración continua ya que permite reducir la deuda técnica derivada de la integración de distintos servicios entre sí. Esta consiste básicamente en intentar integrar los servicios que se disponen a cada paso que se da en la creación de software. En proyectos grandes esto podría ser cada día y en proyectos pequeños cada commit.

---

# Objetivos del proyecto

---

A continuación se indican los objetivos, tanto teóricos marcados por los requisitos como objetivos que se persiguen con el proyecto. También se incluyen ciertos requisitos que no tienen por que ser obvios pero que en la actualidad se esperan de cualquier aplicación web.

## 2.1. Objetivos principales

- Conseguir un sistema actual y escalable.
- Seguir principios de desarrollo actualizados.
- Proporcionar control de usuarios a aquella persona que lo necesite y su proyecto encaje con el que aquí se muestra.

## 2.2. Servidor web y página web

- Arquitectura MVC (Model View Controller)
- Facilidad de uso: Que el diseño sea intuitivo y fácil de aprender a usar.
- Internacionalización: Preparar la aplicación para que este disponible en varios idiomas.
- Sistema responsivo: Que se adecue al dispositivo desde el que se visita.



### 2.3. Servicio de Minería

### 2.4. Proceso de desarrollo

Los puntos siguientes se ven como necesidades fundamentales en cuanto al desarrollo de proyectos actualmente a los que quizá no se de suficiente importancia.

- Sistema para el control de dependencias
- Despliegue de el proyecto
- Sistema de control de versiones
- Tener un proceso de CI (Continuous integration)
- Programar con agilidad

### 2.5. Objetivos personales

- Aprender arquitecturas actuales, las cuales se pueden usar tanto en industria como en academia.
- Avanzar mis conocimientos a partir de los obtenidos en la carrera, sobre todo aquellos que tienen importancia real y quizá no se hayan estudiado suficiente en la carrera
- Profundizar en el entorno de Python, ya que en los últimos años se ha incrementado su importancia tanto para *Data Scientists* como para desarrolladores web, que son las dos profesiones encuentro muy interesantes.

---

## Conceptos teóricos

---

Algunos conceptos teóricos tanto de la parte técnica como de la parte de procesos de software

### 3.1. Deuda técnica

La metáfora de deuda nació como una forma de expresar la diferencia entre el entendimiento del programa (la abstracción) y su implementación [3].

Con el tiempo nació un termino similar la deuda técnica que es la metáfora que expresa el coste que conlleva el dejar código que no tiene un buen diseño en el proyecto. Si seguimos manteniendo el proyecto esta deuda cada vez costará más el arreglarla y tendremos que pagar los intereses [4].

Las razones de incremento de la deuda con el paso del tiempo pueden ser el simple hecho de olvidar como funciona cada linea de código, o la complejidad emergente del acoplamiento de código antiguo con otro nuevo sin cambiar las abstracciones ni desacoplar el código.

Este tipo de deuda se ve como necesaria, según algunos autores [11] se debe incurrir en deuda técnica ya que es beneficioso para el desarrollo. Esto se debe a que a corto plazo es imposible prever cómo se va a desarrollar el proyecto y unos cambios demasiado tempranos pueden ser un mal diseño a medio o largo plazo. La deuda ‘buena’ o beneficiosa se suele considerar como aquella que como mucho dura una semana, una vez existe por más tiempo pasa a ser cada vez más costosa de solucionar o ‘pagar’.

Una característica importante de esta deuda es que una vez dejas de soportar el mantenimiento de un proyecto la deuda técnica desaparece, al contrario que la deuda financiera.

Se puede idealizar la deuda técnica y considerar que todas las semanas en las que se dedica una cantidad de tiempo a solucionarla esta deuda técnica

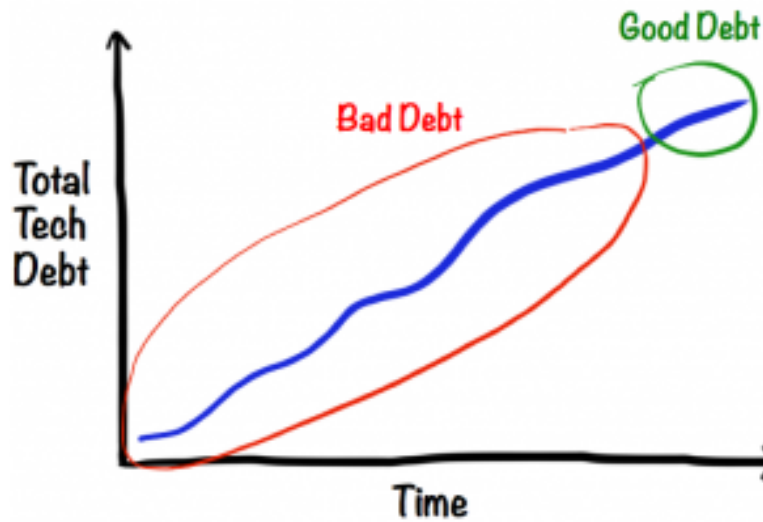


Figura 3.1: Deuda técnica: Beneficiosa y perjudicial. [11]

desaparece, pero esto no corresponde con la realidad. Normalmente la deuda aunque se solucione suele ir incrementando poco a poco debido a la complejidad de un sistema mantenido a lo largo de una gran cantidad de tiempo.

La manera de tener esto en cuenta es tener un ‘techo de deuda’ este techo debería ser lo suficientemente alto como para que no se alcance todos los meses pero no demasiado alto como para que cuando se llegue el proyecto sea directamente un fracaso o demasiado caro de pagar.

La opinión de algunas personas parece ser que se debe llegar al techo cada 6 meses, desde mi propia inexperiencia creo que deberíamos evaluar la deuda a los 4 meses para ver si la re-estructuración del proyecto es viable e intentar buscar un punto de tiempo en el que el pagar la deuda técnica sea algo más simple ya que si esperamos hasta los 6 meses probablemente no podamos elegir un momento óptimo.

Es importante destacar que la deuda técnica tiene otros razonamientos para tener que mantenerse baja: al que menos importancia parece darse es que la deuda técnica es algo difícilmente cuantificable hasta que se refactoriza, si no la reducimos no sabemos cuanta tenemos.

### 3.2. Integración continua (*Continuous Integration*)

Normalmente acortado con las siglas CI, es un método habitual para reducir la deuda técnica. Es la práctica de ejecutar y testear conjuntamente todas

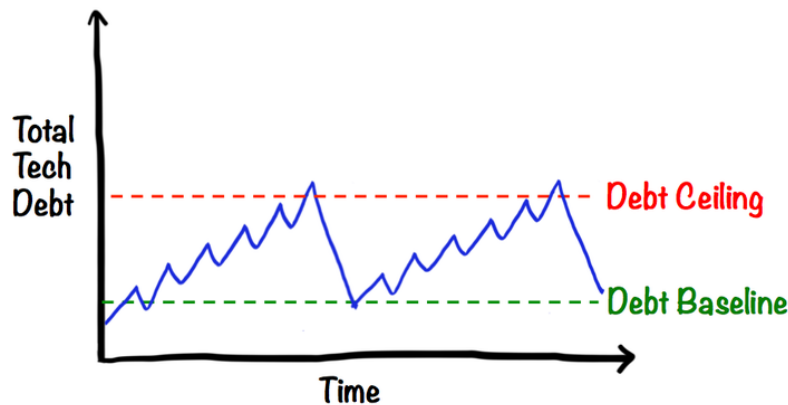


Figura 3.2: Deuda técnica: Techo y base [11]

los servicios que deban ir relacionados una vez se inicie el despliegue en producción. Esto nos asegura que van a funcionar en producción la mayoría de las veces, la minoría habrá problemas por la escalabilidad, errores dado el cambio del hardware, problemas con el rendimiento... Un ejemplo es si ejecutamos java y los test no fallan pero al ponerlo en producción le damos a la JVM más de 200 GB de memoria RAM, esto causa comportamientos inesperados.

### Entrega continua (Continuous Delivery)

Es un término asociado a CI, consiste en un proyecto que dispone de CI y buena cantidad de test de calidad, una vez ya tengamos un sistema a punto podemos empezar a modificar o mejorar funcionalidades añadiéndolas directamente a producción si pasa los test, requiere que hagamos los test casi a la vez que el código. Esto fomenta y premia técnicas como extreme programming (XP) que se basan en TDD (Test Driven Development), una forma de programar que pide que hagamos los test antes que el resto del código.

## 3.3. DevOps

DevOps es un acrónimo inglés de: ‘software **D**evelopment and information technology **O**perations’ es un termino que engloba un conjunto de prácticas de colaboración y comunicación entre desarrolladores software y técnicos informáticos. Los objetivos de esta comunicación y colaboración son una construcción de software más consistente y confiable. Este proceso heredero de las técnicas ágiles se basa en una *cadena de herramientas*. Esta cadena de herramientas es algo que no esta completamente definida pero más o menos la podemos concretar, cabe tener en cuenta que esta cadena cambia según a quien le preguntes.

## La ‘cadena de herramientas’ de DevOps

Esta cadena de herramientas se basa en siete procesos con sus correspondientes herramientas:

1. Plan: Consistente en determinación de métricas, requerimientos... y una vez pasemos de la primera iteración ha de tener en cuenta el feedback del cliente.
2. Creación: Es el proceso de programar y crear el software, las herramientas en este proceso es el software de control de versiones que vayamos a usar.
3. Verificación: Proceso de comprobación de la calidad del software. Normalmente consiste en hacer test de diversos tipos (Aceptación, seguridad...)
4. Preproducción o empaquetación: En esta fase se piden aprobaciones de los distintos equipos y se configura el paquete.
5. Lanzamiento: En este punto se prepara el horario de lanzamiento y se orquesta el software para poder ponerlo en el entorno de producción objetivo.
6. Configuración: Una vez el software esta desplegado toda la parte de la infraestructura y configuración de la misma se incluye en esta categoría, como por ejemplo las bases de datos, configuración de las mismas...
7. Monitorización: Tras entregar el software se mide su rendimiento en la infraestructura objetivo y se mide la satisfacción del usuario final. Se recogen métricas y estadísticas

## 3.4. Microservicios

Los microservicios son una arquitectura y un patrón de diseño, que no se ha inventado en un momento dado, si no que ha surgido como una tendencia o patrón del diseño de sistemas en el mundo real.

Un microservicio es un servicio pequeño y autónomo que puede trabajar junto a otros, es importante que este centrado en hacer una cosa bien.

Esto esta reforzado por el concepto del Principio de responsabilidad única de C. Martin [12] "Juntar aquellas cosas que cambian por la misma razón y separar aquellas que cambian por razones diferentes."

El tamaño de un microservicio es algo muy discutido, pero generalmente se requiere que sea mantenible por un equipo 'pequeño' (6-10 personas), y se pueda reescribir en 2 semanas.

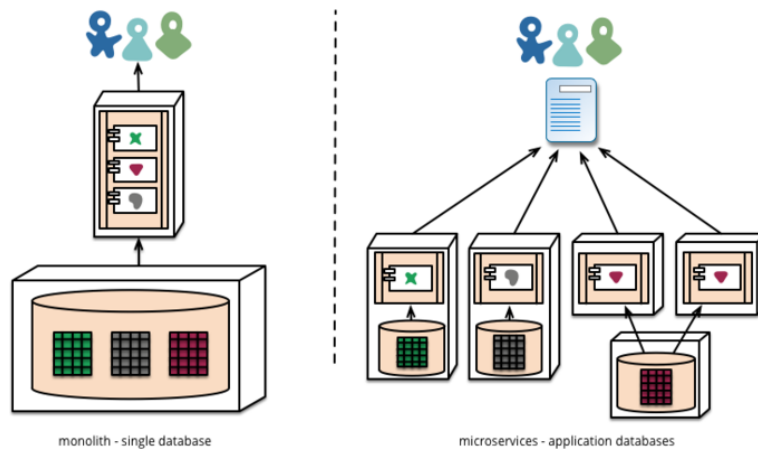


Figura 3.3: Arquitecturas monolítica y microservicios.

Esto se debe conseguir mediante una interfaz de programación de aplicación (API: application programming interface), que permita a los clientes acceder al servicio sin causar acoplamiento, algo más difícil de hacer que de decir.

Los principales beneficios son:

1. Sistemas heterogéneos: Facilita usar distintas tecnologías en distintos sistemas para usar la mejor herramienta en cada ocasión.
2. Resiliencia: Si falla un microservicio el resto del sistema se puede mantener levantado, aislando los problemas y facilitando la alta disponibilidad.
3. Escalabilidad: En caso de que una parte del sistema necesite mayor número de recursos podemos crear una nueva instancia sin cambiar el funcionamiento del resto del sistema.
4. Facilidad de despliegue: Los cambios se mantienen más contenidos.
5. Replazabilidad: Un microservicio que pasa a ser obsoleto puede ser reemplazado o eliminado en vez de tener que mantenerlo por que al eliminarlo se rompe todo el sistema, algo que no debería pasar pero que es común en los sistemas legados.
6. Test a nivel de servicio: Cada microservicio se puede testear por su cuenta propia de manera que aunque el sistema aumenten su complejidad podamos controlar cada microservicio por separado.

Estos beneficios por supuesto vienen con desventajas:

1. Distribución: Raramente se van a ver microservicios en una arquitectura que no este distribuida, esto hace que sea más difícil de programar y tengamos que tener en cuenta otros errores.
2. Seguridad: Es más difícil asegurar la seguridad de un microservicio ya que debemos protegernos contra más puntos de vulnerabilidad, a no ser que estemos ejecutando en un clúster seguro.
3. Complejidad de operación: Hace falta un equipo de operaciones con madurez de manera que se puedan redistribuir los microsistemas regularmente.
4. Test a nivel de sistema: Los tests a nivel de sistema incrementan algo su complejidad ya que vamos a tener que usar microservicios falsos como mocks o similares casi siempre a no ser que estemos en un sistema pequeño.

Por ultimo vale la pena hablar del coste en productividad, esto no es una ventaja ni un perjuicio, es un *trade-off*, un intercambio.

En un sistema monolítico tenemos menos coste en productividad hasta que llegamos a un punto crítico en el que la complejidad aumenta demasiado o necesitamos escalar.

En los microservicios tenemos menos coste a la hora de seguir iterando aumentando la complejidad de un sistema pero a cambio tenemos mayor barrera de entrada hasta tener el primer prototipo funcional y los desarrolladores tienen que aprender cómo programar bajo esta arquitectura.

---

# Técnicas y herramientas

---

En esta parte de la memoria se detallan tanto técnicas y herramientas como metodologías y librerías usadas.

## 4.1. Metodología

Al no haber una metodología concreta que se adapte al trabajo de una sola persona se ha buscado un conjunto de metodologías que se pudiesen conjuntar para dar una metodología conjunta más aplicable a la situación particular que se tiene.

De **DevOps** se ha extraído la toolchain (cadena de herramientas) para minimizar riesgos totales del proyecto y se ha buscado reducir la deuda técnica nacida de la necesidad de desplegar el producto y no estar preparado para ello, algo que resulta ser más habitual de lo que debería ser.

De **Scrum** recogemos las herramientas como los sprint y las reuniones cada sprint tanto para concretar el siguiente sprint como para hacer una retrospectiva del anterior.

Se ha estudiado también **extreme programming** (XP) y se ha visto que no es viable para el proyecto por la cantidad de esfuerzo que requiere para una sola persona hacer el trabajo de dos como es el **pair programming**.

De todos los métodos ágiles se ha adquirido la mentalidad de no intentar planificar todo desde el principio e intentar planear las cosas con el conocimiento adquirido en cada iteración. Se intenta decidir en todo lo más tarde posible para intentar conseguir el máximo conocimiento posible para mejorar las decisiones.

Se ha decidido que la recopilación de información de los sprints y este tipo de documentación se va a hacer en un tablero **kanban**, la herramienta que nos proporciona este sistema va a ser **Zenhub**, se usa por la facilidad y el nivel de integración que nos da con otros sistemas también elegidos.



## 4.2. Patrones de diseño

Se ha intentado usar los patrones de diseño para conseguir una arquitectura mejor y más simple para nuevos desarrolladores. También hay que tener en cuenta que no todos los patrones tienen sentido en un lenguaje de programación como python por su naturaleza dinámica.

### Model-View-Controller

El patrón modelo vista controlador nos ayuda a simplificar mucho la estructura de los archivos de una página web o aplicación que dependa de vistas. Esto es especialmente beneficioso cuando se introduzca a un miembro nuevo al proyecto ya que permite conocer rápidamente como funciona mucha cantidad de código, de manera que si necesitase realizar cambios lo tuviese fácil.

## 4.3. Control de versiones

El control de versiones es una necesidad hoy en día tanto en sistemas compuestos de uno o múltiples desarrolladores, esto nos permite poder dejar ciertas features que empezamos a implementar y se dificulten sin terminar para seguir con otras que den más valor al cliente y luego continuar con las dificultades o dejarlas de lado según beneficie al cliente.

Este solo es un ejemplo de las ventajas entre las que se cuentan: recuperación de versiones estables, visualización de los cambios que han dado resultado a un bug...

El sistema de hosting lo usaremos para facilitar el acceso a la información contenida en el sistema, también, al ser externo (no estar en el mismo ordenador que se use) nos servirá de sistema de copias de seguridad.

Usaremos **git** por razones de documentación y conocimiento ya adquirido, aunque existen otros sistemas como subversion.

Las alternativas principales de sistema de hosteo de git son: Bitbucket, Github, Gitlab.

Se ha elegido Github por una mezcla de razones históricas con razones de integración con otros sistemas como el sistema kanban (Zenhub).

Históricamente se conocía Github y se sabe que los proyectos open source no tienen ninguna limitación, la fácil integración con otros sistemas como **travis** y **slack** si hiciese falta.

## 4.4. Integración continua

La integración continua es el método por el cuál intentamos integrar todos nuestros productos continuamente para ver si funcionan correctamente en conjunto. Nos va a ayudar a minimizar el riesgo de fallo del proyecto sobre todo si se mantiene en desarrollo un largo periodo de tiempo.

La herramienta que se va a usar es **travis**, esto se debe a la gran documentación, facilidad de integración con git y gihub, y otras integraciones que nos ayudaran con otras secciones. Otra ventaja es que al ejecutar tu build y tests en sus servidores no te tienes que preocupar de casi nada.

Otras opciones que se podrían usar son Jenkins que como ventaja es Open Source y tiene gran cantidad de plugins.

## 4.5. Quality Assurance

Dado que no tenemos departamento de QA como deberíamos para llevar una toolchain como la especificada en DevOps usaremos herramientas automatizadas, que aunque no tengan la calidad de una revisión humana es lo mejor que disponemos.

La herramienta que se ha elegido es **CodeClimate**, se ha usado por que no ha sido demasiado complicado el incluir esta herramienta en la linea de producción que ya teníamos (Github+travis). Se ha intentado usar SonarQube pero no ha sido tan sencillo. (Se intento introducir dentro de la toolchain automática, no manualmente).

## 4.6. Tests

Los tests se han realizado con **pytest**. Esta herramienta es muy parecida a unittest (la herramienta de tests en la biblioteca standard de python). Tiene ventajas, facilita el debug al decirte exactamente que ha fallado y con que valores, y los métodos como setup y teardown. Se integró esta herramienta con travis.

Otras librerías pueden ser unittest o nose2.

## Recubrimiento

La forma más sencilla para ver el recubrimiento de un test en Python es coverage, tiene varias opciones como report para salida en consola o html para un html con el cubrimiento bien señalado. Se consiguió integrar esta herramienta con travis, pytest y CodeClimate de manera que se ejecuta con la integración continua.

## 4.7. Dependencias

Para el control de dependencias (seguridad, últimas versiones y licencias) se ha usado VersionEye, integrado con Github, esta herramienta se adhiere mediante un webhook a Github y nos dice si se ha descubierto alguna brecha de seguridad en nuestras herramientas, cuales son sus últimas versiones y si las estamos usando y si las licencias del proyecto son compatibles con la que tenemos.

## 4.8. Comunicación

La comunicación se ha hecho de diferentes maneras: Email, Slack y de manera física. Email se ha usado para casi toda la comunicación a distancia. Slack se ha usado para integrar herramientas y notificar sobre estas. La comunicación física es el medio que más se ha usado, esto no se debe a ningún motivo en particular simplemente ha sido el medio más natural para todos.

## 4.9. Documentación

La documentación se ha basado en dos sistemas: en código y fuera de código.

### En código

En el código se ha usado la documentación recomendada por la comunidad de python en el PEP 287 [5], este recomienda usar **reStructuredText** (rst).

Se han seguido las guidelines (guías) de un equipo de programadores de la comunidad de python: pocoo [10], esto es para poder usar Sphinx (generador de documentación) para transformar esas cadenas de documentación en una documentación tanto en html o en Latex si hiciese falta.

### Fuera de código

Para la documentación que se esta leyendo se ha usado la plantilla oficial de la ubu para crear documentos Latex aunque se ha considerado usar rst al igual que en el código ya que si hiciese falta se podría transformar a Latex o a html siendo mucho más flexible. No se ha hecho por que no existe una plantilla y la barrera de entrada es algo alta.

## 4.10. Editor de texto

Se ha mirado tanto en editores de texto más simples como vim, atom, sublime... Estos cuentan con suficientes plugins que acaban siendo una IDE.

Las IDEs también se han mirado y se han considerado tanto Spyder, Lynclipse como Pycharm.

De todos estos no hay ventajas en usar unos u otros ya que todos acaban teniendo la misma capacidad gracias a plugins, se ha decidido usar Pycharm por que era la única IDE que todavía no se había probado.

## 4.11. Bibliotecas

Se ha usado Flask como microframework ya que sirve tanto para implementar el patrón MVC como para crear un sistema altamente escalable.

Para facilitar ciertas funcionalidades como control de usuarios, seguridad... se han usado las extensiones: flask-babel y babel (internacionalización), flask-login (control de usuarios), flask-oauthlib y oauthlib (uso de oauth simplificado), flask-sqlalchemy y psycopg2 (acceso a la base de datos), flask-wtf y WTForms (Formularios html) y flask-bcrypt y Bcrypt para seguridad.

Como podemos ver muchas de estas extensiones de flask tienen el mismo nombre pero con flask añadido, esto se debe a que son wrappers de la librería en cuestión pero facilitados para inicializarse con flask (generalmente añaden un constructor y un constructor lazy).

Se ha usado el modulo paramiko para simplificar el uso de SSH.

---

# Aspectos relevantes del desarrollo del proyecto

---

[No se sabe si faltan parrafos de introduccion y definiciones de conceptos o así esta bien]

## 5.1. Idea e inicio

Se eligió un proyecto con relación con el deep learning ya que existía un interés previo tanto a nivel personal como a nivel profesional. La gran versatilidad de este tipo de redes, ya sea en clasificación con los modelos profundos, compresión con los autoencoders o incluso generación de imágenes con las revolucionarias GAN, esta claramente demostrada, ahora queda buscarles nuevos usos que supongan una aplicación práctica más directa y útil.

A nivel personal la importancia que tienen las redes neuronales en el mundo actual hizo que surgiese una gran curiosidad. Un ejemplo de la aplicación de este tipo de tecnología podrían ser los coches automáticos o aplicaciones surgidas últimamente, como la viral durante un par de días FaceApp.

A nivel profesional el deep learning se ha posicionado como uno de los algoritmos que mejor resultados han dado en muchos campos, esto abre tanto puertas profesionales y posibilidades de trabajo en industria como posibilidades de investigación y estudio en academia.

## 5.2. Formación

El proyecto requería una serie de conocimientos de los que no se disponía en un principio, como es normal en cualquier proyecto no trivial. Algunos de los conocimientos necesarios se aprendieron durante la carrera pero por supuesto

en un campo tan grande y cambiante como es la informática la gran mayoría de tecnologías se tuvieron que investigar y aprender por cuenta propia.

Lo necesario para llevar a cabo el proyecto se expondrá en las siguientes secciones, se intentaran omitir detalles que parezcan triviales o de muy poco interés para el lector de manera que la lectura no sea demasiado pesada.

Antes de hacer nada más lo principalmente necesario fue el conseguir un entorno con integración continua.

Esto se presenta como una dificultad bastante grande ya que el único libro conocido sobre el tema [ref] aunque parece fiable según algunas personas es mucho más largo de lo que debería, y teniendo en cuenta que esta centrado para java se ha decidido intentar evitarlo.

Los materiales alternativos a este libro han sido generalmente entradas de blogs y hablando con gente que lo ha usado (aunque quizá no suficientemente experimentados). [<https://martinfowler.com/articles/continuousIntegration.html>] [<https://www.atlassian.com/agile/continuous-integration>]

Otras cosas que se han tenido que aprender han sido tensorflow[ref] y metodologías de uso de git[gitflow].

Tensorflow se estudió con los tutoriales oficiales relevantes para las partes a los que se quería dar uso. Aunque quizá no fuese necesario también se revisó la tensorflow con[youtube ref] para comprobar tanto la escalabilidad como las posibilidades más punteras en servidor y de distribución de los modelos en otros dispositivos.

Gitflow se siguió conforme a la guía en datasift. [<https://datasift.github.io/gitflow/IntroducingGitFlow.html>]

### 5.3. Página web y versionado de bases de datos

No se conocía el desarrollo web con python ni con flask. Para adquirir el conocimiento sobre esto se usó los tutoriales [explore flask] y [megatutorial de flask].

Hay que tener cuidado porque el primer tutorial está en python2. Esto generalmente no da casi ningún problema ya que flask se ocupa de muchos de los problemas de versión pero a la hora de usar SQLAlchemy los hash de las contraseñas en python2 se pueden guardar como string pero en python3 se deben guardar como objetos de bytes.

Una de las lecciones más importantes aprendidas de estos tutoriales es la importancia del versionado del modelo de la base de datos. Esto se hace para poder migrar de una versión a otra de forma automática.

Es una manera bastante fácil de mantener la base de datos en buenas condiciones. Esto se debe a que si al hacer alguna migración falla, se cancela y

no perdemos datos, de manera que si alguna migración falla podemos añadir lógica al como migrar para facilitar el proceso.

A parte de poder hacer versiones o 'commits' a mano, podemos hacer versionado automático de la base de datos lo que infiere a partir de los modelos el cambio que ha ocurrido en la base de datos. Esta opción no se ha trabajado ya que no se ha llegado a necesitar.

### Alembic: primeros pasos

Alembic nos permite de forma efectiva versionar las bases de datos de la misma manera que versionamos el código. Para poder empezar a usar alembic en un proyecto debemos ir a la carpeta del proyecto y ejecutar el comando:

```
alembic init alembic
```

Este comando creará tanto la carpeta alembic como el archivo alembic.ini, ambos colgando de la ruta desde donde hayamos ejecutado el comando.

En la carpeta se almacenan las versiones y scripts de migración entre ellas. El archivo ini tiene la configuración de alembic y para que sepa donde tenemos la base de datos tendremos que cambiar la ruta de la base de datos a donde tengamos la nuestra. Esta ruta se guarda en el parámetro sqlalchemy.url

```
sqlalchemy.url = driver://user:pass@localhost/dbname
```

Para crear una revisión (commit inicial en este caso) debemos usar el comando:

```
alembic revision -m "message"
```

Cabe destacar que esto influye sobre todo en el mantenimiento de sistemas en producción ya que si no esta en producción los datos son más fáciles de recuperar y se pueden recurrir a alternativas menos ortodoxas (hacerlo a mano, usar scripts hechos a mano...).

## 5.4. Heroku y transición a contenedores

Tras tener la versión básica de la página web se decidió hacer un despliegue en heroku, una de las páginas que nos permiten publicar nuestra página web dinámica.

Heroku en concreto funciona con contenedores pero nos quita gran parte del peso del aprendizaje, la versión gratuita nos limita a un contenedor (donde dejamos nuestro proyecto como si fuera un ordenador normal) y nos deja un segundo contenedor limitado a una base de datos postgresql con unos limites de tamaño.

Tras conseguir completar el despliegue en heroku y tener que integrar tensorflow en el proyecto, para facilitar la integración, el despliegue en otros sistema y el mantenimiento del software se decidió hacer los contenedores de forma explicita. La alternativa elegida fue docker, por ser la opción más madura.

Para aprender a usar docker se usó la documentación oficial [link] y el libro [pack docker]. [hacer una explicación basica de docker al estilo de la de alembic?]

## 5.5. Docker compose y microservicios

Docker compose sirve para organizar nuestros contenedores en despliegues concretos de manera que se puedan usar en cualquier momento, por ejemplo podríamos tener varios docker compose con distintas configuraciones según nuestras necesidades, tanto para distintos servicios de alojamiento como para distintas configuraciones de despliegue. Esto último es algo bastante complicado y que si nos propusiesemos hacerlo tendría que ser con antelación o refactorizaciones decentemente grandes.

El aprendizaje de docker compose fue sobre todo mediante la documentación oficial aunque el libro [cita al packt] también se usó. [explicar docker compose al estilo de alembic si/no]

Se puede desplegar con docker compose pero con una gran limitación y es que no nos permite usar varios ordenadores de la manera que los servicios de orquestación nos lo permiten.

Los microservicios como arquitectura se conocieron debido a su popularidad actual, muchos blogs han hecho artículos sobre ellos y algunos newsletter como el de O'Reilly o el de Nginx les han dado mucha importancia últimamente, adjuntando tutoriales y conferencias con las ultimas noticias.

## 5.6. Orquestración

Los conocidos como orquestradores nos permiten, como su nombre indica, orquestrar servicios entre varios ordenadores lo cual nos permite elegir configuraciones más versátiles y acordes a nuestras necesidades.

Los orquestradores son docker swarm, kubernetes y Openshift. [explicar kubernetes... si/no?]

Docker swarm fue el servicio que se intentó usar debido a la existencia de secretos, que son ficheros con contraseñas o archivos que no deberían ser públicos. Estos secretos no están en docker compose, pero si en swarm aunque el tamaño del enjambre sea uno, que es equivalente a compose, lo que parece una estrategia comercial más que un diseño bien planteado.



Usar docker swarm dio gran cantidad de problemas y tras una investigación online la opinión general es que ya está listo para entornos de producción, la experiencia que podemos relatar es que swarm da problemas extra, que son difíciles de diagnosticar, quizá sea un caso concreto o que se hizo algo de manera incorrecta pero tras bastante tiempo perdido sin ser capaces de avanzar se decidió continuar con otras partes del proyecto.

## 5.7. Conclusiones

### Gitflow, scrum y metodologías pensadas para equipos

Gitflow es un sistema complejo y que requiere conocimiento medio de git para poder llevarse a cabo. Realmente no se necesita para proyectos tan pequeños como los de una sola persona y el trabajo que requiere realmente se vio que no merecía la pena. Aunque parece beneficiosa para equipos no se ha podido comprobar y cabe considerar que existen críticas de la metodología en cuestión [ref].

Scrum también es parecido, se ha visto como funciona en otros entornos y es sin duda beneficioso sobre todo para equipos de tamaño mediano. Una cosa que sí que merece la pena de scrum es el tablero kanban, aunque solo seas una persona merece la pena como forma de tener claras que partes del proyecto están hechas hasta que punto.

### Tensorflow, versiones provisionales y tecnología punta

Tras aprender tensorflow el verano pasado y no seguir usándolo tan activamente durante el primer cuatrimestre se presentó un problema, tensorflow al estar en versiones de desarrollo al sacar la versión 1.0 se perdió parte del conocimiento adquirido, la api cambió, algunos de los scripts dejaron de funcionar y se desarrollaron librerías más maduras y de más alto nivel, dejando otra parte del conocimiento adquirido parcialmente obsoleto.

Como ya se ha comentado docker swarm también dio problemas, la suposición es que se hizo algo incorrecto en docker o docker compose que hizo que swarm no funcionase correctamente. Al ser tecnología punta no hay tantas preguntas respondidas en stack overflow o foros similares sobre el tipo de problemas que nos encontramos.

Docker compose también dio problemas, más ligeros que docker swarm, con las configuraciones de red. Suponemos lo mismo pero estos sí que se pudieron arreglar.

## **Microservicios**

[ Reescribir: Esta muy bien cuando quieres proporcionar un servicio muy ampliable pero para los proyectos de menos de 3 servicios separados no se ven, sobre todo pensado para proyectos a largo plazo no a corto plazo, añade demasiado trabajo que hacer sin proporcionar suficientes beneficios ]

---

# GNU Free Documentation License

---

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.

`<http://fsf.org/>`

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

## Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document “free” in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book.

We recommend this License principally for works whose purpose is instruction or reference.

## 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “**Document**”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “**you**”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “**Modified Version**” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “**Secondary Section**” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “**Invariant Sections**” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “**Cover Texts**” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “**Transparent**” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to

text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “**Opaque**”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “**Title Page**” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “**publisher**” means any person or entity that distributes copies of the Document to the public.

A section “**Entitled XYZ**” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “**Acknowledgements**”, “**Dedications**”, “**Endorsements**”, or “**History**”.) To “**Preserve the Title**” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

## 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are

reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

### 3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

### 4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on.

These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

- K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.



## 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

## 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

## 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in

an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

## 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

## 9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this

License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

## 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

## 11. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

## ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with ... Texts.” line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

---

## Bibliografía

---

- [1] Martin L Abbott and Michael T Fisher. *The art of scalability: Scalable web architecture, processes, and organizations for the modern enterprise*. Pearson Education, 2009.
- [2] André B Bondi. Characteristics of scalability and their impact on performance. In *Proceedings of the 2nd international workshop on Software and performance*, pages 195–203. ACM, 2000.
- [3] Ward Cunningham. *Debt Metaphor*, 2009.
- [4] Martin Fowler. *TechnicalDebt*, 2003.
- [5] David Goodger. *PEP 287*, 2002.
- [6] Michael Hausenblas. *Serverless Ops*. O’Reilly Media, 2016.
- [7] Mark D Hill. What is scalability? *ACM SIGARCH Computer Architecture News*, 18(4):18–21, 1990.
- [8] *Image-net Team*. *Image-net*.
- [9] *Serverless Team*. *Serverless*.
- [10] *Image-net Team*. *Image-net*.
- [11] Henrik Kniberg. *Good and Bad Technical Debt*, 2013.
- [12] R.C. Martin. *Agile Software Development: Principles, Patterns, and Practices*. Alan Apt series. Pearson Education, 2003.