

# Trabajo Fin de Grado

## Infraestructura de personalización y monitorización de crawlers basada en Docker

Autor

Íñigo Alonso Ruiz

Director

Francisco Javier López Pellicer

Grado en Ingeniería informática  
Departamento de Informática e Ingeniería de Sistemas  
Escuela de Ingeniería y Arquitectura  
2016

# Infraestructura de personalización y monitorización de crawlers basada en Docker



**Universidad**  
Zaragoza

---

# Infraestructura de personalización y monitorización de crawlers basada en Docker

---

## Resumen

Algunos proyectos requieren la creación de arañas web o crawlers para obtener datos concretos de la web. Estas arañas suelen ser construidas enfocadas para un uso concreto y su configuración es bastante compleja y costosa en lo que en tiempo se refiere. El objetivo de este Trabajo de Fin de Grado es el desarrollo de un sistema de creación, personalización, y monitorización de crawlers basado en contenedores virtuales Docker definidos mediante un pequeño lenguaje de configuración o DSL (Domain Specific Language) sencillo y de un sistema de persistencia de datos para la información recolectada por los crawlers.

El sistema está desarrollado para poder ser utilizado para uso individual, o colectivo. Puede ser gestionado a través de línea de comandos, dando posibilidad a un uso más rápido a usuarios más expertos, o vía web, donde el sistema será gestionará la posibilidad de ser usado por varios usuarios a través de una interfaz usable y sencilla.

Las funcionalidades sobre los crawlers van desde su creación, configuración, monitorización de su estado, control del mismo e incluso un buscador e indexación de la información recogida propio.

A pesar de que un sistema de crawling completo pueda ser muy costoso de crear, gracias a Docker y su reutilización de partes de sistemas ya construidos, la creación es casi inmediata, aparte de otras muchas ventajas que ofrece como su portabilidad y ligereza (tamaño en memoria) respecto a las máquinas virtuales convencionales.

Así pues, a través de un desarrollo incremental guiado por pequeñas iteraciones, y dirigido por pruebas (inspirado en la conocida aproximación TDD - Testing Driven Development) se ha ido construyendo un sistema en constante evolución, unificando varias tecnologías para conseguir como resultado un sistema potente que posibilita la construcción casi inmediata de instancias de sistemas de crawling.



## DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./D<sup>a</sup>. Íñigo Alonso Ruiz,

con nº de DNI 73013097B en aplicación de lo dispuesto en el art.

14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster)  
Grado \_\_\_\_\_, (Título del Trabajo)

Infraestructura de personalización y monitorización de crawlers basada en  
Docker

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 3 de mayo de 2016

Fdo: \_\_\_\_\_

# Agradecimientos

Quiero agradecer el esfuerzo de muchos **profesores** que intentan de verdad que sus alumnos aprendan, que van más allá de un simple empleo y se implican en ellos, que van más allá de enseñar la materia y enseñan valores morales, dan apoyo, ánimo y mantienen una relación alumno-profesor más cercana.

Quiero agradecer a aquellos profesores que hacen que te intereses por su asignatura, que los alumnos no solemos agradecer estos detalles y son estos profesores los que hacen que los alumnos tengan mucha más motivación por lo que hacen y que vayan más allá del trabajo de clase.

Agradecer a estos profesores como entidad de profesor y como personas, porque la universidad es algo diferente gracias a ellos. Quisiera nombrar a alguno, que no son todos, porque tan solo he recibido clase de unos pocos profesores, ojalá poco a poco y con mucho esfuerzo, haya más profesores de este tipo. También agradecer a los profesores que se esfuerzan porque una comunidad así sea posible:

*Francisco Javier López Pellicer, Javier Campos, José Manuel Colom, Diego Gutiérrez Pérez, Francisco Javier Fabra, Enrique Torres, Luis Manuel Ramos, Juan Domingo Tardós, José Luis Briz...*

Agradecer como no, a Francisco Javier López Pellicer la gran ayuda que ha sido en este trabajo sacrificando más tiempo del que suele ser normal en este tipo de proyectos para que el resultado del mismo sea de un nivel alto.

También cómo no, quiero dar las gracias a mi madre y a mi padre, que son los que me han financiado la carrera y mis dos grandes pilares de apoyo, y que, sin ellos, yo no sería para nada lo que soy ahora mismo y nunca podré agradecerles lo suficiente. Y gracias a todas las personas que son verdaderos amigos, gracias.

## Contenido

1. Introducción.....	8
1.1 Objetivo y alcance del proyecto .....	8
1.2 Motivación.....	9
1.3 Contexto .....	9
1.4 Tecnologías y sistemas relacionados.....	10
1.5 Metodología y técnicas.....	11
1.5.1 Metodología.....	11
1.5.2 Herramientas utilizadas .....	11
2. Arquitectura del sistema .....	12
2.1 Decisiones de diseño .....	14
3. Diseño del sistema.....	15
3.1 Butler .....	15
3.1.1 Butler DSL.....	16
3.1.2 Validador .....	17
3.1.3 Comandos .....	18
3.1.4 Configuración .....	19
3.1.5 Indexador y buscador.....	20
3.2 Sistema web.....	21
3.2.1 Integración de Butler .....	21
3.2.2 Servidor .....	22
3.2.3 Cliente.....	22
3.3 Evolución y decisiones de diseño.....	23
4. Tecnologías usadas, la magia de Docker.....	25
5. Problemas encontrados .....	28
5.1 Problemas previamente conocidos .....	28
5.2 Problemas inesperados .....	29
6. Validación y pruebas del sistema .....	30
7. Resultados.....	31
8. Conclusiones.....	32
8.1 Trabajo futuro .....	32
8.2 Valoración personal.....	33
9. Trabajo y esfuerzo realizado y gestión del proyecto .....	34
10. Bibliografía.....	39
Anexos.....	41
Anexo A. Butler en detalle. Guías y esquemas. ....	41
Especificación del DSL .....	43
Guía de usuario.....	46
Anexo B. Sistema web .....	51
Guía de usuario.....	51
Anexo C. Validación .....	56
Anexo D. Indexación.....	58
Anexo E. Instalación Docker.....	59

## Índice Figuras

Figura 1. Diagrama del sistema completo.....	12
Figura 2. Reparto de horas por mes del proyecto.....	34
Figura 3. Reparto de horas por tarea del proyecto.....	35
Figura 4. Número de commits realizados en el proyecto.....	35
Figura 5. Cantidad de código subido a GitHub durante el proyecto.....	35
Figura 6. Reparto de horas en el mes de febrero por tareas.....	36
Figura 7. Reparto de horas en el mes de marzo por tareas.....	36
Figura 8. Reparto de horas en el mes de abril por tareas.....	37
Figura 9. Reparto de horas en el mes de mayo por tareas.....	37
Figura 10. Reparto de horas en el mes de junio por tareas.....	38
Figura 11. Diagrama de actividad del validador y constructor de Butler.....	41
Figura 12. Inicialización de Butler.....	46
Figura 13. Ejecución de comandos de Butler.....	50
Figura 14. Registro de un usuario.....	51
Figura 15. Creación de un proyecto.....	52
Figura 16. Creación de una imagen del proyecto 'aaaa'.....	53
Figura 17. Contenedor 'one' ejecutándose.....	54
Figura 18. Contenedor 'one' en estado pausado.....	54
Figura 19. Ejemplo de test de Butler.....	47
Figura 20. Diagrama de secuencia de la indexación.....	48

# 1. Introducción

Existe muchísima información en internet, y esta información además de crecer a una velocidad desorbitada, es cambiante. Para poder buscar o encontrar información, se necesita algún mecanismo de recolección de datos y de búsqueda sobre ellos, al primer mecanismo, se le llama crawler o araña web, al segundo, motor de búsqueda.

Los crawlers recolectan información por toda la web, pero, existen muchos métodos y algoritmos para la recolección de esta información, ya sea porque se quiere optimizar el tiempo, se quiere recolectar solo la información más importante (según unos criterios) o porque solo se quiere buscar un tipo de información (focused crawler).

El sistema que se ha desarrollado pretende simplificar el proceso de creación de estos crawlers dando cabida a todo tipo de ellos dejando la personalización de este al usuario, pero simplificando muy significativamente el proceso de creación del mismo y habilitar un mecanismo de control y monitorización sobre ellos.

## 1.1 Objetivo y alcance del proyecto

El objetivo de este proyecto es realizar un sistema que permita crear crawlers personalizados de una forma rápida y sencilla para reducir costes monetarios y temporales en esta tarea, realizando una infraestructura que sea capaz y soporte la monitorización de una gran cantidad de crawlers. El proyecto se ha enfocado a un desarrollo incremental y modular, así poder tener módulos separados con funcionalidades útiles (como un Shell, un DSL) que puedan ser utilizadas por si solas.

El alcance del proyecto se establece a través de unos objetivos mínimos que el sistema debería cumplir a la finalización del proyecto, que son los siguientes:

- Un DSL (Lenguaje específico del dominio) con su correspondiente librería (guía de uso y ejemplos) que permita especificar, de manera sencilla, un trabajo de crawler.
- Un sistema validador para el DSL.
- Una aplicación (un *Shell* propio) que permita gestionar desde línea de comandos dicho DSL y realizar un conjunto de acciones sobre los sistemas de crawling creados:
  - Creación y personalización de crawlers
  - Operaciones de control y monitorización sobre el crawler
- Un sistema web, tanto backend como frontend que permita el manejo de este sistema a nivel de usuario.



## 1.2 Motivación

La motivación de este proyecto, es en parte la situación actual del mercado que se explicará en 1.3 Contexto, realizar un sistema de código abierto que facilite la tarea de creación de crawlers personalizados reduciendo significativamente el coste temporal y monetario.

Al tener este sistema ya no solo es que puedas tener un sistema de personalización de crawlers de una forma rápida y muy fácil, sino que tomando como base este proyecto, se puede crear infinidad de ellos enfocados a otro tipo de sistemas que no sean el crawling.

Pero más allá del ámbito empresarial y/o utilidad, la mayor motivación de este proyecto es también personal. Como alumno de la especialidad de computación, tenía curiosidad por los sistemas webs, poder realizar uno completo por mí mismo aprendiendo varias tecnologías y frameworks en el proceso, aunque llevase más tiempo de la cuenta para realizarlo. Una motivación de aprendizaje, un reto para mí mismo y por el hecho de experimentar, y hacer un sistema más enfocado a software, dado que en mi especialización no he podido indagar en ese aspecto.

## 1.3 Contexto

Hoy en día no existen muchos sistemas de creación y personalización de crawlers, dado que es un tema complejo de manejar. Existen compañías que controlan el ámbito de los crawlers o buscadores web, pero todas ellas tienen debilidades, ya sea porque no se centran en información específica, no permiten personalizaciones detalladas u otros defectos que permiten existan necesidades a abordar. Dentro de los sistemas existentes que predominan en la construcción de sistemas de crawlers personalizados, más en relación a lo que se enfoca este proyecto, son, por ejemplo, 80Legs, DataMiner, ScrapingHub etc. [W19]

Cuando las empresas necesitan este tipo de sistemas, o se ven forzados a crear un crawler desde cero, o a buscar otro método para la recogida de esta información, como los que se acaban de nombrar, pero estos ofrecen poca variedad en la personalización y además no son baratos.

Así, un sistema de código abierto como este, sencillo de utilizar, que gestiona los recursos de forma eficiente y rápido de instalarse en cualquier sistema, sería utilizado por muchísimos usuarios, sobretodo empresas guiadas más en la búsqueda de información concreta a través de los *focused crawlers*.

## 1.4 Tecnologías y sistemas relacionados.

El sistema creará los crawlers utilizando Docker. Docker es una tecnología nueva, pero muy potente, así pues, ahora mismo, muchas empresas están apostando por ella, subiendo sus sistemas encapsulados en contenedores Docker a [W11] (plataforma de repositorios Docker), y los sistemas de crawling no se quedan atrás, por ejemplo, [W9], tiene tanto una imagen Docker subida al Dockerhub, como el fichero Dockerfile y las instrucciones para los usuarios que quieren personalizar *Nutch* (el sistema de crawler que este proyecto soporta por defecto) con Docker.

Relacionado con sistemas más allá de la encapsulación de Nutch no existe nada actualmente, así que además de a través de tutoriales, los ejemplos útiles sobre los que poder aprender o extraer algo interesante son estas encapsulaciones de sistemas en Docker que básicamente son la puesta en marcha de un sistema operativo y la descarga del sistema a utilizar, su configuración e instalación a través de comandos de la API de Docker.

En referencia a proyectos de Nutch, se pudo extraer el set-up del mismo en proyectos Docker, pero tuvo que modificarse respecto a funcionalidades internas de Nutch (scripts) y extraer ciertas herramientas como Solr y HDFS para las primeras versiones. Sobre todo estos proyectos sirvieron para tener una base mínima sobre la que más tarde, poder construir el sistema de personalización, el cual modifica propiedades y configuraciones de Nutch y de Docker.

También se utilizó la documentación oficial de Nutch para saber su funcionamiento, sus posibilidades de configuración qué hay que editar si se quieren ciertos cambios en el sistema.

En la parte de Docker, todo lo que se encontró sirvió para probar sobre ellos o tomarlos como fuente de inspiración y aprendizaje, y se tuvo que realizar el sistema de Docker y el sistema de crawling casi desde cero, pues no había nada que encajase con lo que se buscaba. Docker, al ser una totalmente nueva, y parte de la base del sistema, se tuvo que invertir tiempo en conocer el funcionamiento básico y útil para el proyecto y ver qué posibilidades daba a parte de la funcionalidad base.

Por el lado de la realización de un Shell propio, se investigó sobre sistemas que acelerasen el proceso de creación de este, y *Spring Framework*, tiene un proyecto llamado *Spring Shell* que permite con cierta facilidad la creación de comandos en una Shell propia.

De la parte del sistema web, hay infinita variedad de ejemplos de sistemas web de código abierto. Para este sistema, se ha utilizado la documentación de los frameworks utilizados para la misma y tutoriales existentes de estos frameworks como de AngularJS o SpringMVC.

## 1.5 Metodología y técnicas

La metodología realizada, se ha enfocado hacia un desarrollo incremental para facilitar su desarrollo e ir definiendo y concretando aspectos en cada iteración.

### 1.5.1 Metodología

Respecto la realización del sistema, se ha seguido un desarrollo incremental, evolucionando el sistema de un sistema simple, aumentando su potencia y funcionalidad a medida que se iban añadiendo módulos. Para asegurar el continuo funcionamiento del sistema, se siguió un desarrollo guiado por TDD. El desarrollo guiado por pruebas, también llamado TDD, es una técnica de desarrollo basada en establecer unas pruebas que el sistema debe de pasar satisfactoriamente, a continuación, implementar el software para que pase la pruebas, y una vez esto ocurre, refactorizar el código y mejorarlo para mejorar la calidad del mismo.

El desarrollo se ha realizado mediante pequeñas iteraciones, de unas dos o tres semanas con tareas a realizar en cada una de ellas, y la coordinación entre director y alumno se ha realizado a través de una entre cada una de estas iteraciones y a través de dos portales/aplicaciones web: *Github* para el seguimiento de las tareas y *Slack* para una comunicación más inmediata.

### 1.5.2 Herramientas utilizadas

Las herramientas utilizadas en este proyecto han sido bastantes, debido sobre todo al hecho de que una de las motivaciones del proyecto era aprender nuevas tecnologías.

- Parte del DSL y Shell: Java como lenguaje, Lucene para la indexación y motor de búsqueda, Nutch como sistema de crawling principal, Spring Shell para el Shell propio, Junit para test y gradle para manejo de dependencias.
- Para el sistema web: Bootstrap como framework CSS, AngularJS como framework JavaScript, Spring MVC como framework para backend y gradle para manejo de dependencias.

Se ha utilizado también Github como sistema de control de tareas y milestones/iteraciones y Slack como medio de comunicación del proyecto.

## 2. Arquitectura del sistema

El sistema desarrollado está formado por dos grandes módulos: Butler y el sistema web.

La base del sistema es Butler, el cual contiene la mayor parte de la lógica de negocio y la innovación del proyecto, pero se apoya en el sistema web el cual hace que su manejo esté al alcance de cualquier usuario y termina de estructurar el sistema centralizando y uniendo todos los sistemas que lo rodean.

El sistema final queda dividido en 5 grandes partes que se pueden apreciar en la *figura 1*.

- Web Services. Construidos modularmente y construidos para poder realizar un sistema con varios servidores explicados en 3.2.2 Servidor
- Base de datos. El ya nombrado sistema de ficheros con una base de datos relacional para tener accesible la información de control de cada usuario explicado en 3.2.2 Servidor.

La base de datos está desacoplada del servidor, es decir, a esta accede tanto Butler (al sistema de ficheros) como los servicios web (tanto al sistema de ficheros como a la base de datos relacional) y pueden acceder tantos sistemas como se quieran. Para esto, la base de datos, cuanto más accesible y robusta sea, mejor.

- El cliente. El cliente web, tan solo interacciona con los servicios web. La parte del cliente explicada en 3.2.3 Cliente.
- Butler. Encapsula la funcionalidad del sistema explicado en detalle en 3.1 Butler.
- Contenedores Docker. Donde se alojan los sistemas de crawling. Estos ahora mismo, se ejecutan en el mismo servidor donde se alojan los servicios web, pero ya se ha explicado la forma inmediata de poder tener ejecutándose los contenedores en varias máquinas externas. explicado en 4. Tecnologías usadas.

A continuación, se muestra un diagrama de la arquitectura del sistema final.

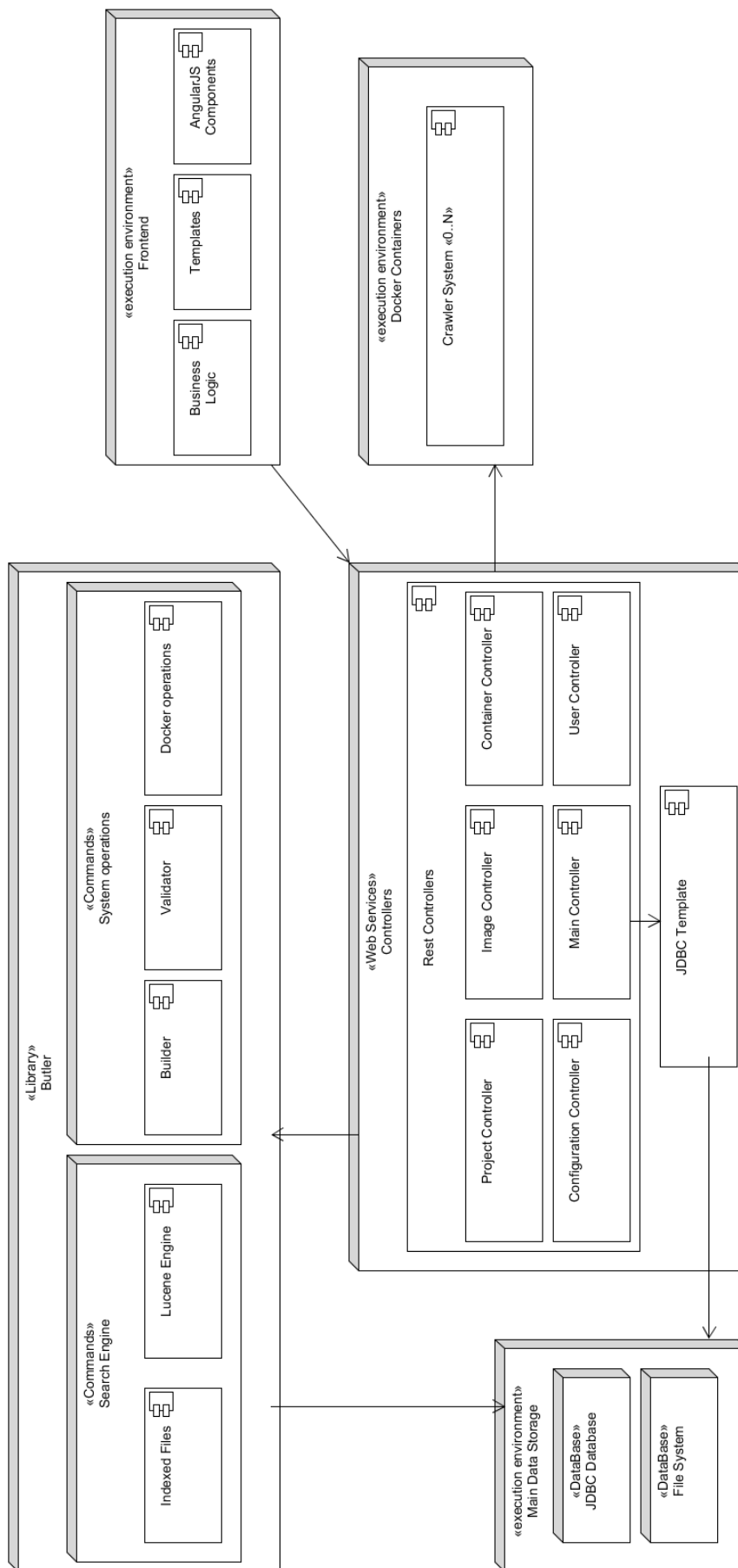


Figura 1: Diagrama del sistema completo.

## 2.1 Decisiones de diseño

Las decisiones de diseño arquitecturales se han ido realizando según se avanzaba en el proyecto.

Lo primero que se tuvo era Butler, un módulo con la mayoría de la funcionalidad de todo el sistema, el core del proyecto. A partir de aquí es cuando se necesitaba realizar un diseño que explotase su potencial buscando la escalabilidad.

Solo con Butler, el sistema se basaba para el uso de tan solo un usuario y en un ordenador, para poder aumentar el uso del sistema se pensó en la utilidad de un sistema web para dar cabida a diferentes usuarios.

El sistema web debía ser escalable y modular, y se debía encontrar un sistema de ficheros y base de datos que soportase las necesidades del sistema, lo cual hizo que este módulo fuera externo a el sistema web y así pudiera haber varios servidores conectados al mismo sistema de datos.

La parte del cliente se realizó como una aplicación *one-page*, pero el requisito más importante respecto al diseño era que el estado estuviera contenido en el cliente, para poder realizar un diseño RESTful.

Del sistema, lo único que faltaba por definir era casi, lo más importante, cómo y dónde se va a ejecutar los contenedores Docker. Estos contenedores son los que consumen recursos y lo que mejor se ha de administrar.

La primera versión del sistema en este aspecto, que es la que está implementada es que se ejecuten todos los contenedores en la máquina donde se esté ejecutando el servidor, pero la implementación futura y final en este aspecto sería tener almacenadas todas las máquinas accesibles para utilizar y ejecutar vía *ssh* la ejecución de los contenedores Docker, ya sea por una distribución equitativa de consumo de recursos entre todas las máquinas o a elección del usuario.

## 3. Diseño del sistema

El sistema se compone en dos grandes módulos a alto nivel. Butler que es la esencia del proyecto y el sistema web que lo integra. En este apartado se explica qué es, cómo está construido y como funciona cada uno de ellos, así como la evolución del sistema y las decisiones de diseño realizadas.

### 3.1 Butler

Butler es el sistema que integra toda la funcionalidad relacionada con la creación y personalización de los sistemas de crawling.

Es un sistema construido con una metodología modular, compuesto por varias partes, cada cual se encarga de una función específica y cuya funcionalidad puede ser aumentada fácilmente.

Lo primero de todo es explicar y responder unas preguntas simples e introductorias al sistema.

¿Qué es Butler?

Butler es una aplicación de Shell (línea de comandos) que contiene comandos propios y específicos.

¿Qué hace o puede hacer exactamente Butler?

Butler facilita la creación y personalización de sistemas de crawling y su posterior control ofreciendo un amplio abanico de operaciones al usuario.

Butler también alerta de cualquier fallo posible de la aplicación o de las tecnologías que usa, tanto previa ejecución gracias a los test integrados que contiene como en ejecución gracias a la continua comprobación de que las operaciones a realizar sean posibles, incluso que las aplicaciones externas que usa, como Docker, estén disponibles

¿Cómo?

A través de un DSL propio, un pequeño lenguaje, que, a través de un fichero de configuración sencillo, en formato *yalm*, permite especificar las características de todo el sistema, desde el sistema operativo de la máquina donde va a correr el sistema de crawling, como qué sistema de crawling quieres (ej. *Nutch*) hasta especificaciones del crawler, como sin ir más lejos, la/s semilla/s.

### 3.1.1 Butler DSL

Un DSL (Domain Specific Language), es un pequeño lenguaje creado para un sistema o ámbito específico (Especificación del DSL y explicación en el Anexo A).

Butler hace uso del DSL para poder especificar de una forma fácil, sencilla y rápida las características del sistema de crawling que se desea crear a través de un fichero de configuración, dado que sino, para realizar esto, el usuario tendría que editar a mano, ficheros de configuración específicos de cada sistema o realizar los cambios costosos pertinentes.

Un sistema de crawling, normalmente tiene muchas propiedades o características que el usuario puede cambiar. Por dar un número de ejemplo redondo, pongamos que pueden ser mil.

Para una primera versión del DSL se han escogido las propiedades más importantes y útiles enfocadas a un uso tanto empresarial como individual. Así, estos son los aspectos que se pueden especificar sobre su configuración que se aplicarán desde el momento en el que se empieza a ejecutar.

- Configuraciones mínimas obligatorias
  - El nombre y la versión del sistema de crawling.
  - Las semillas que servirán como base al sistema.
  - El número de rondas o iteraciones que el sistema permanecerá en ejecución.
  - Si la extracción de información del crawler se debe realizar en cada ronda o tan solo al terminar su ejecución.
  - El tipo de información a extraer.
- Configuraciones opcionales
  - Tipo o modo de cola.
  - Información respecto a timeouts o tiempos de delay.
  - Restricciones sobre la información a recoger.

Para que este sistema sea realmente configurable, se permite añadir funcionalidades o comportamientos propios al usuario. Esto es posible dejando la posibilidad de añadir al sistema plugins para que sean ejecutados en el sistema de crawling.

Estas especificaciones, sirven para todos los sistemas de crawling compatibles con Butler, es decir, el usuario especifica con el DSL las características que desea, y Butler, por debajo, dependiendo si se ha especificado *Nutch* o *Heritrix*, implementa los cambios pertinentes.

También se debe especificar obligatoriamente el sistema operativo y su versión sobre la que se quiera que el sistema se ejecute.



Para que todo esto funcione, el sistema tiene *templates* por defecto, tanto de Docker como cada uno de los sistemas de crawling para poderlos modificar según las especificaciones del usuario y poder construir un crawler a medida.

El sistema tiene un adaptador, el cual va construyendo poco a poco los ficheros necesarios. Este adaptador funciona como un coordinador, el cual llama a los constructores necesarios con la información necesaria y los coordina para que no existan problemas.

De este modo, para añadir un nuevo sistema de crawling a Butler, tan solo habría que implementar el constructor propio para ese sistema.

### 3.1.2 Validador

Para realizar un sistema con control de fallos, para completar el DSL se necesita como mínimo un validador, que además de, por su puesto, validar si un fichero de configuración es sintácticamente y semánticamente correcto, ayudar al usuario qué error es el que está haciendo que exista el problema y decirle dónde está, es decir, *feedback* al usuario para que pueda entender y comprender por qué da error y corregirlo lo más rápido posible.

Para ver más en detalle la estructura del validador y constructor del validador y el constructor, ver el *Anexo A* y la *figura 2*.

El validador tiene una estructura también modular. Existe un pequeño validador por cada campo que se puede especificar en el DSL, y cada uno realiza las comprobaciones pertinentes, como, por ejemplo, si es obligatorio si está presente o validar que el tipo o valor de dato introducido sea correcto.

También es el encargado de asegurar que los plugins que recibe el sistema tengan un formato correcto y que no le falten ficheros o especificaciones a este.

Cada uno de estos validadores, es encapsulado por otros que validan un conjunto de especificaciones de un mismo conjunto, en el caso de esta versión de Butler dos, uno para el sistema de Docker y otro para el sistema de crawling, y otro un nivel superior que encapsula a estos dos.

Los ficheros de configuración están en formato *yalm*. Para mapear la información que contienen, utilizar una librería externa. [W13]

Una vez validado, se pasará a la construcción de los ficheros de salida necesarios para el crawler. La arquitectura del constructor es similar a la del validador, con la diferencia de, que respecto al sistema de crawling, utiliza el patrón de diseño Adapter, para poder variar su ejecución según el sistema que se escoja.

### 3.1.3 Comandos

La funcionalidad de Butler se centra en los comandos del Shell. El Shell se ha construido a partir del proyecto de *Spring Shell*, de *Spring*, el cual ofrece un sistema Shell, configurable (Guía de uso y especificación en el Anexo A).

Existen dos tipos de configuraciones:

- Configuraciones de la consola, las cuales se basan tanto en el aspecto del Shell al ejecutarse como en el modo de ejecutarse. Las configuraciones de este tipo, se ha realizado lo básico, aunque no trivial.
- Configuraciones de comandos: Estas configuraciones son las referentes a la creación de cada comando, los argumentos que tiene cada uno y sus características. *Spring Shell* facilita esta labor a través de las anotaciones de java entre otras cosas.

Para abarcar una amplia funcionalidad, se han implementado varios comandos, cada uno enfocado a realizar una sola acción respecto al sistema, se podrían dividir en tres grupos:

- Comandos de creación
  - Crear una configuración. Es decir, dada unas especificaciones, un fichero de configuración de acuerdo con el DSL creado, un comando que cree todos los ficheros necesarios para el sistema.
  - Creación de la imagen Docker. Esto, sería el equivalente, para quien no esté familiarizado con Docker, a la puesta a punto del sistema operativo y descargar/instalación de todas las herramientas que se necesita para montar un sistema de crawling. Aunque pueda sonar muy costoso, Docker puede realizarlo en tan solo un segundo.
- Comandos de eliminación o parada
  - Parada del sistema de crawling. Se necesita un comando para poder llegar a parar el crawler por si fuera necesario.
  - Parada del contenedor Docker. El contenedor Docker es similar a una máquina virtual, así pues, es útil tener un comando que de opción a parar el contendor
  - Pausa del contenedor Docker. Pausar el contendor es algo más suave que parar, es más recomendable, así como parar el contendor es un símil de apagar, el símil de pausar sería congelar la ejecución.
  - Eliminación del contenedor Docker. Un comando de eliminación es necesario tanto por no ocupar recursos de memoria.
  - Eliminación de la imagen Docker.

- Comandos de funcionalidad del crawler
  - Arranque del contenedor. Este comando coge la imagen de Docker, e instancia un contenedor con el contenido de esta imagen y lo arranca, así se tendrá un contenedor virtual para cada crawler.
  - Arrancar el crawler. Como su nombre indica, arranca el sistema de crawling.
  - Obtener el estado del crawler. Sirve para saber si el crawler ha terminado ya, si está en ejecución, o en cambio ni siquiera ha empezado a ejecutarse.
  - Obtener el estado del contenedor. Obtiene el estado del contenedor de Docker, el cual puede estar en ejecución, pausado o parado.
  - Obtener información sobre el crawler. Sirve para saber por ejemplo cuantos links se llevan visitados o cuantas iteraciones se han ejecutado.
  - Indexar el contenido del crawler. Indexa el contenido recogido por el crawler para poder ser utilizado posteriormente de una forma eficiente, se verá más en detalle en 3.1.5 Indexador y buscador.
  - Buscar sobre el contenido del crawler. Realiza búsquedas en el contenido indexado.

### 3.1.4 Configuración

Para un desarrollo más rápido y eficiente, todo proyecto debe tener una buena configurabilidad.

En este proyecto hay dos partes desde las que se puede configurar el sistema

- Ficheros de configuración. En estos se especifican las rutas donde están los recursos. En el caso de este sistema, los recursos son los *templates* por defecto que se utilizan para configurar los sistemas de crawling y de Docker y todo recurso necesario para las pruebas de validación del sistema.
- Beans. Las *beans* son usadas por *Spring*, son simplemente objetos (referentes a lenguaje de programación) que definen una estructura o comportamiento de este. Así, se pueden definir objetos de un tipo/clase, que se tomaran por defecto al ser instanciados previa declaración de una anotación específica de java.[W8]  
Butler utiliza este sistema para configurar, desde la Shell y sus propiedades, hasta los validadores nombrados en 3.1.2 Validador

### 3.1.5 Indexador y buscador

Una vez se ha recopilado la información, esta, ha de pasar un proceso para que las búsquedas y el manejo de ella sea eficiente en el tiempo.

Este procesado se llama indexación, y existen herramientas que ayudan a no tener que implementar todo el proceso desde cero. En este sistema, se ha utilizado *Lucene*. [W17]

Se han creado dos índices, uno para el contenido descargado por cada URI y otro para el nombre de la URI.

Debido al enfoque de este sistema, el analizador de *Lucene* que se ha escogido ha sido el *EnglishAnalyzer*, el cual realiza todo el procesamiento (*Stopwords*, *tokenización*, *normalización*, *lematización*, *stemming*...) en inglés.

Para las búsquedas sobre este índice, también se utiliza *Lucene*, pero antes, se realiza un pre-procesado, como puede ser, por ejemplo, limpieza de signos ortográficos.

La indexación se puede realizar tanto dentro de cada contenedor Docker como a petición del usuario guardando el índice en donde el sistema, Butler, esté ejecutándose.

Así, debe haber una coordinación mínima, entre estos dos posibles lugares a la hora de realizar la búsqueda. El encargado de solucionar esto es Butler, que, al realizar la búsqueda, sabe cuál de los dos índices está más actualizado y realiza la búsqueda sobre este, eliminando el otro índice por estar desactualizado.

Desde los contenedores Docker, para que la indexación automatizada se realice con la misma metodología que a petición del usuario, una de las acciones que realiza el contenedor en su set up, es la descarga interna de Butler para utilizar el mismo código que Butler y así no duplicar tampoco código.

Butler tiene entonces, un sistema propio de indexación y búsqueda, el cual te permite, desde la Shell, entre otras cosas, saber el número de resultados encontrados o poder pedir un número limitado de resultados.

Para más detalle en el sistema, ir al *Anexo D*.

## 3.2 Sistema web

Para que el sistema pueda ser utilizado a través de un sistema más accesible por todo el mundo, la mejor solución es llevarlo a la web, y ha si ha sido.

A pesar de que no es el sistema principal ni protagonista del proyecto, ha llevado el mismo tiempo de implementación que Butler debido a las nuevas tecnologías que se han tenido que aprender para su realización y que ha tenido que integrarse Butler en el sistema, pero gracias a este sistema web, se ha dotado de potencia y usabilidad al sistema.

El sistema está compuesto por varios elementos

- Butler
- Parte del cliente
- Parte del servidor
- Sistema de datos (Base de datos y de ficheros)
- Contenedores Docker en ejecución

El sistema está pensado para que el sistema de datos esté compuesto por una Base de datos relacional y otra que no lo sea o un sistema de ficheros.

Los contenedores Docker en esta primera versión se ejecutan en la máquina donde se aloja el servidor, pero el sistema está pensado para poder ser ejecutados en máquinas externas, y, por último, el servidor, ahora mismo es único, pero, perfectamente podría estar compuesto de varios nodos compartiendo la misma base de datos, dada el modularidad del sistema y por una apuesta por REST para mantener el estado fuera de la parte del servidor.

### 3.2.1 Integración de Butler

Butler es una aplicación Shell, pero para transformarlo en un sistema web, se ha tenido que pensar tanto en la mejor forma de hacerlo como en la estructura del sistema que se iba a montar.

La mejor solución encontrada, fue crear un archivo formato *jar*, que encapsulase a Butler y que permitiera ejecutar los comandos de Butler a través de la ejecución de Butler. Así, el servidor podría operar sobre Butler de una forma sencilla, y, el servidor forma parte del mismo proyecto que Butler, así al compilar a través de *gradle* todo el sistema, el servidor tendrá siempre la última versión de Butler.

Para comodidad del usuario se amoldó los parámetros de Butler a una estructura más enfocada al sistema web, estructurando por proyectos.

### 3.2.2 Servidor

El servidor es la parte del sistema donde se centraliza toda la información, además de ser el lugar donde los contenedores Docker van a ejecutarse.

Además de integrar la funcionalidad de Butler, el servidor ofrece alguna más.

Principalmente se compone de dos partes:

- Web services. Para los servicios web, el servidor hace uso del *Spring MCV* para aligerar el desarrollo y apostando por un enfoque a servicios REST.

Estos servicios, en el proyecto se dividen en varios controladores, uno por cada entidad necesaria, es decir, hay un controlador para usuarios, otro para proyectos, otro para los contenedores etc., y cada uno de estos controladores contiene los servicios necesarios minimizando las dependencias.

Esto se hace para, además de mejorar la calidad del código, poder tener de forma casi inmediata varios servidores distribuidos

- Base de datos. Para guardar toda la información que se va generando se utiliza una base de datos de tipo relacional.

También se tiene un FS (File System) donde se almacenan todos los ficheros de configuración que se van generando para la construcción de las imágenes y contenedores Docker y los sistemas de crawling. Se estructuran por usuario y proyecto. Este FS, pasará a ser un HDFS (*Hadoop Distributed File Sytem. Un sistema de archivos distribuido y escalable de Apache*) en las futuras versiones del sistema.

### 3.2.3 Cliente

La parte del cliente está desarrollada enfocada hacia una *one-page website*.

El uso de *AngularJS* facilita la estructuración de código, agiliza el desarrollo y mejora la experiencia del usuario, y esto, unido a *Bootstrap, framework* de CSS, aportan lo necesario para tener un diseño y UX que agrade a casi cualquier usuario y que pueda ser usado desde cualquier tipo de plataforma.

El desarrollo *one-page* es similar al convencional, usando las directivas de *AngularJS*, tan solo hay que indicar donde se quiere insertar el código de diferentes templates, y esto unido al *data-binding* que ofrece, dependiendo del estado de la aplicación (la cual está en el cliente, ya que se ofrecen servicios REST), se puede mostrar unos contenidos u otros al usuario de una forma muy sencilla.

El *data-binding* además ofrece una muy buena UX y ofrece más posibilidades para poder crear contenido de alta calidad.

La única contra que tiene *AngularJS*, es la pronunciada curva de aprendizaje, y llegar a dominar este framework, aunque sea a un nivel básico, no es tan trivial como otros.

### 3.3 Evolución y decisiones de diseño

Durante la realización del proyecto, el sistema ha ido tomando forma, evolucionando poco a poco desde una aplicación muy sencilla. Lo único que tiene que ver con el resultado final es la idea de partida.

A través de una continua evaluación de posibilidades, decisiones de diseño, mejoras y aprendizaje de nuevas herramientas, el sistema ha ido tomando forma hasta el resultado final.

Al principio, la idea del proyecto era clara, lo que se quería hacer, pero no exactamente el cómo.

Se quería realizar un proyecto lo más profesional posible y enfocado al uso de tecnologías, pero al principio, antes de usar *Spring Shell*, Butler empezó siendo un conjunto de scripts escritos en *bash*. Cada uno tenía una funcionalidad básica usando Docker y aún son parte del proyecto, aunque no tienen toda la funcionalidad que tiene Butler, sí que se pueden crear crawlers muy básicos y extraer la información.

En la fase de documentación sobre Docker y ejemplos existentes con Nutch, se iban evaluando y buscando librerías para leer ficheros *yalm*, y de librerías o proyectos para pasar de la versión de scripts a una integrada con un futuro sistema de validación y construcción del crawler en *java* ya que para un sistema como el que se quería construir, se necesitaba una base sólida y segura.

Spring Shell no fue la única opción, una vez descartado el uso de scripts, se encontraron varias librerías que permitían realizar comandos en java, pero, al ser un proyecto de Spring, la integración con otras herramientas sería más sencilla.

Sobre esta etapa de documentación, se tuvo que cambiar de dirección en lo que la construcción de los sistemas de crawlers se refiere. Se intentaba construir un sistema de crawling dentro de Docker el cual incluía su propio HDFS (Hadoop Distributed File System) y el motor de búsqueda de Solr, pero, tras pensar bien sobre la arquitectura que se quería y evaluar diferentes opciones, la elegida fue encaminar todo a una centralización de la información recogida por los crawlers hacia un futuro servidor que se encargará del control y acceso a cada uno de la información de cada contenedor.

Una el proyecto iba tomando forma y se veía más claro las herramientas a utilizar, se creó el sistema de validación siguiendo un patrón de diseño Singleton [W14] [W15], y se especificó las configuraciones mínimas se pretendían soportar en la primera versión del DSL (Mirar de arriba abajo todas las posibles configuraciones que ofrece Nutch y escoger las más importantes).

Cuando se tuvo por fin la especificación y desarrollo del DSL completo, (que más tarde aún se incrementó algo más dado que es un desarrollo incremental), era momento para empezar a desarrollar Butler con *Spring Shell*, y evaluar nuevas posibilidades relacionadas con el diseño como ¿Qué comandos se necesitan? (*apartado 3.1.4*) ¿Es viable y lógico tener Solr o HDFS dentro de los contenedores Docker? (No, *apartado 1.3*) ¿Se necesita un patrón de diseño Adapter en el Validador? (Sí, y además un coordinador, *apartado 3.1.2*).

Una vez convertida toda la aplicación en una aplicación *SpringBoot*, se necesitó empezar a poner un poco de orden, se integró un sistema de *logger* por toda la aplicación, se documentó todo hasta ahora en varias wikis y se paró la implementación una iteración para dejar paso a una búsqueda y limpieza de errores.

Una vez se tuvo Butler completo (más tarde, según se iban necesitando más comandos, se iban añadiendo), se empezó a crear el sistema web desde cero, un reto personal para aprender. Las herramientas elegidas para esta tarea fueron *Spring MVC*, *AngularJS* y *Bootstrap*, las dos primeras explicadas en 4. Tecnologías usadas.

Esta opción se empezó a desarrollar previo descarta de *Spring remote Shell*, dado que a pesar de que hubiera aumentado las posibilidades y potencia de la Shell ya implementada, no era lo que se buscaba exactamente, y un sistema web, permite el acceso desde cualquier lado que tenga acceso a internet y distribuir el sistema.

A pesar no ser la base del sistema, el desarrollo web y la integración de Butler en él llevó casi el mismo tiempo que la implementación de Butler mismo.

El desarrollo de la parte del servidor y la del cliente fue simultáneo. Se iba desarrollando funcionalidad a funcionalidad, integrando Butler en el servidor, con una primera versión simple en ambas partes, y en el momento en el que todo el funcionamiento era perfecto, se pasó a mejorar la estructura implementada en la parte del servidor simplificando el código, y en la parte del cliente, enfocándose más a mejorar el diseño y la usabilidad.

Referente a las decisiones de diseño en esta parte, fueron bastante simples. El servidor es en enlace de todos los demás elementos y centraliza la información del sistema. La necesidad de existencia de usuarios era obvia, y las funcionalidades a ofrecer eran las contenidas en Butler con ciertos matices. Se decidió que el diseño fuera *REST* y lo más simple y modular posible.



## 4. Tecnologías usadas

La motivación principal de este proyecto, era la de aprender algunas de las tecnologías que están en auge ahora mismo en diferentes campos del software, y así crear un sistema compacto y potente que combina varias tecnologías para una desarrollar y madurar la faceta del aprendizaje y adaptación a tecnologías nuevas.

La tecnología sobre la cual gira el proyecto es Docker. [W1]

Docker es un proyecto (de software libre) que permite automatizar el despliegue de aplicaciones dentro de contenedores virtuales.

Estos contenedores de Docker se podrían definir como máquinas virtuales ligeras y portables, un proceso que encapsula la aplicación que queremos ejecutar y sus dependencias.

El que sean muy ligeros es debido a su arquitectura, diferente al de las máquinas virtuales corrientes, un contenedor Docker no contiene todo un sistema completo, sino únicamente aquellas librerías, archivos y configuraciones necesarias para desplegar las funcionalidades que contenga.

El contenedor puede ser desplegado en cualquier otro sistema (que soporte esta tecnología), con lo que se ahorra el tener que instalar en este nuevo entorno todas aquellas aplicaciones que normalmente usamos.

En el caso de los desarrolladores, el uso de Docker hace que puedan centrarse en desarrollar su código sin preocuparse de si dicho código funcionará en la máquina en la que se ejecutará.

Además, Docker da facilidades para controlar los cambios que se hagan en los contenedores a través de una API/comandos de más alto nivel que facilitan la gestión de estos.

La idea es crear una imagen base, sobre la que realizar cambios para configurarla. Una vez hecho los cambios, mediante la API de Docker, se crea la imagen a usar. Esta imagen contiene únicamente las diferencias que hemos añadido con respecto a la base. Docker se encarga de darnos la base que comparten las imágenes y de acoplar los diferentes cambios de cada imagen que hemos creado.

En conclusión, Docker es una herramienta que puede empaquetar una aplicación y sus dependencias en un contenedor virtual muy ligero, fiable, fácil de gestionar y desplegar, portable y flexible. [W3]

Dentro de Docker (el proyecto) existen varios de ellos que ayudan a convertir las aplicaciones que usan Docker en sistemas autosuficientes, escalables y muy potentes como *Docker Compose*, *Docker Engine*, *Docker Machine*, *Docker Registry*, *Docker Swarm*, *Kinematic*, *Docker Hub*, *Docker Cloud* y *Docker Datacenter*.

Para Butler, Docker es la clave de su potencia y escalabilidad.

Docker le confiere a Butler el poder ser ejecutado en cualquier sitio, el poder desplegar los crawlers donde sea y coordinarlos fácilmente gracias a su API.

Gracias al uso de las cachés de Docker y el funcionamiento de las imágenes que se construyen sobre una imagen base, en el momento en el que Butler crea el primer sistema de crawling, con el sistema operativo, la descarga e instalación de las herramientas necesarias para que todo funcione, todas las siguientes veces, este proceso no se tiene que realizar, y la construcción del sistema es casi instantánea, teniendo que escribir en la imagen tan solo las diferencias de la personalización de este, en el peor caso siendo, la copia de varios plugins al sistema de crawling nuevo.

Con estas capacidades de Docker y la funcionalidad de Butler, se puede llegar a crear un grupo de crawlers bastante amplio funcionando en muy poco tiempo.

Recordando que el uso de recursos se minimiza y reduce de una forma drástica por la estructura interna de Docker, estos sistemas de crawling abarcarán muchos menos recursos de lo normal.

Para la futura versión de Butler que permita distribuir los contenedores en diferentes máquinas, tan solo habría que tener instalado Docker en estas máquinas y tener acceso a ellas, y el funcionamiento sería idéntico, tan solo usando *ssh*.

Una vez creados los contenedores Docker, el manejo de estos es trivial, existiendo comandos de borrado, ejecución, estado, y otras funcionalidades desde la API de Docker.

Referentes a otras tecnologías usadas en el sistema, se podría hablar de *Spring MVC*, *Spring Shell*, *AngularJS*, *Lucene* y *Nutch*.

- *Spring MVC*. Es un framework que agrupa varios proyectos que facilitan el desarrollo de aplicaciones web en *Java* en el lado del servidor. Ofrece infinidad de herramientas relacionadas, por ejemplo, con el mapeo de las peticiones entrantes o como otras relacionadas con las bases de datos. Te permite integrar bases de datos internas a tu aplicación, con funciones básicas predeterminadas para realización de consultas y una *API* que te ofrece realizar queries a más alto nivel y sin necesidad de crear tú la conexión con la base de datos.

Como este tipo de herramientas ofrece muchas otras como colas de mensajería, herramientas para conexiones seguras, etc. [W7]

- *Spring Shell*. Uno de los proyectos de Spring y ya nombrado anteriormente, ofrece un proyecto que incluye una Shell propia y facilita la creación de nuevos comandos haciendo muy sencillo integrar una aplicación *Java* en línea de comandos.

A grandes rasgos, el desarrollador solo se tiene que preocupar de, por cada comando que quiere insertar, especificar el nombre del comando, los argumentos que quiere que tenga, y el tipo y obligatoriedad de cada uno de ellos.

A pesar de ser una nueva tecnología que se ha aprendido en el proyecto, no ha sido mucha complicación, puesto que debido al ser bastante intuitiva y estar en *Java*, ha facilitado mucho el desarrollo. [W4]

- *AngularJS*. Es un framework para JavaScript mantenido por *Google*, lo cual da bastante seguridad al programador de que algo bueno tiene que tener. AngularJs se basa en la utilización de directivas propias, en el data-binding y en la idea de los controladores. Es uno de los más difíciles de aprender, pero una vez tienes las bases se pueden hacer maravillas de una forma muy organizada, estructurada y rápida.

En resumen, se podría decir que hace uso del ámbito (referente a lenguajes de programación) para compartir el máximo de información entre lo que está dentro de un mismo controlador, y conectar esa información en tiempo real (data-binding). Con sus características, hacer una aplicación *one-page* es bastante sencillo, y, así pues, este proyecto se aprovecha de ello. [W5]

- *Lucene*. Es una librería de Apache enfocada a motores de búsquedas. Para su utilización no hace falta ser un experto en recuperación de información ya que te proporciona clases que indexan y recuperan información a alto nivel, teniendo que especificar únicamente el tipo que quieres que sea, si un modelo probabilista, si booleano, si vectorial, o combinaciones existentes entre ellos (como la que viene por defecto).
- *Nutch*. Es un sistema de crawling desarrollado por Apache. En este proyecto es el único soportado por el momento para la personalización del crawler. Lo que es la construcción básica viene hecha con varios plugins y funcionalidades que permiten poder ejecutar un crawler básico y poder obtener información de él. Existen ficheros de configuración como *nutch-site.xml* para poder sobrescribir las funcionalidades por defecto. [W6]

## 5. Problemas encontrados

Durante la realización del proyecto, se han encontrado algunos problemas los cuales se han solucionado ya sea buscando otros caminos o echando más horas, los retos están para superarlos.

### 5.1 Problemas previamente conocidos

- Desconocimiento de tecnologías usadas y desarrollo en un continuo aprendizaje (previo aprendizaje de lo básico), sobretodo desconocimiento de Docker y AngularJS.

Para solucionar esto, antes de aprender la tecnología me documenté de ella y escogí los aspectos necesarios para la realización del proyecto, así aprender solo la base y los aspectos más avanzados que necesitaba el proyecto así como la realización de tutoriales.

- Falta de costumbre en el desarrollo de aplicaciones web, sobretodo en la parte de cliente.
- El amoldarme a seguir patrones de diseño, a los cuales en la rama de computación no se les da tanta importancia.
- El diseño del sistema completo, el integrar todas las partes del sistema de la mejor forma posible explotando y aprovechando cada tecnología al máximo.
- Falta de costumbre de implementación de sistemas tan grandes en comparación con lo que he hecho hasta ahora, en estos casos el desarrollo incremental es una muy buena opción.

Para poder superar estos problemas, me he apoyado en el director del proyecto para que me aconsejase y me guiase en aspectos como los patrones de diseño, así como en las decisiones más importantes, las cuales según avanzaba el proyecto, y me iba sintiendo más cómodo iba necesitando menos ayuda.

## 5.2 Problemas inesperados

- Encontrar una forma de poder ejecutar los comandos de Butler, sin iniciar la Shell, no fue nada trivial dado que Spring Shell no está pensado para esto.

Investigué si había algo parecido hecho por internet pero no había nada ya que Spring Shell no está hecho para esto, así que la integración de Butler se realizó tal cual se explica en 3.2.1 Integración de Butler.

- Funcionalidades puntuales web, como upload/download de ficheros.

Debido a la falta de costumbre del desarrollo web, no había tenido problemas que suelen surgir aunque sean fáciles y cuando los haces una vez ya no tienes problemas con ellos de nuevo. Estos problemas que suelen ser tonterías, quitan bastante tiempo hasta que consigues solucionarlo a base de probar distintas formas de realizarlo.

- Encontrar las herramientas idóneas para este proyecto entre todas las existentes.

Investigando a través de internet, buscando diferentes herramientas para realizar una misma función y comprando las ventajas, inconvenientes, como se acoplaría cada una al sistema, y qué ofrece cada una.

- Amoldar Nutch a sin Solr ni Hadoop con Lucene respecto a las necesidades del sistema.

Debido a que el sistema de ficheros final es único, uno para todos los sistemas de crawler, en las primeras versiones, se descartó de Solr y realizar a través del sistema web y Lucene un buscador propio. A demás como las primeras versiones no se tiene Hadoop, se tuvo que realizar un indexador y buscador con Lucene que se amoldase a las especificaciones del sistema.

Realizado como se explica en 3.1.5 Indexador y buscador, se abordó este problema con guías de Lucene y la experiencia previa sobre esta tecnología.

## 6. Validación y pruebas del sistema

Para que el sistema sea lo más estable y sin errores posibles, tanto en versiones finales como durante su desarrollo, se han de seguir unas pautas y metodologías que ayuden a que esto sea posible.

Durante el desarrollo del código, se debe intentar que existan las menos dependencias posibles, para que cuando se cambie algo en un módulo no afecte a los demás. Para esto, es aconsejable un previo análisis del sistema y un buen diseño, además de seguir patrones de diseño que ayuden a esto.

También seguir un desarrollo guiado por TDD es muy aconsejable, teniendo pruebas de que ciertas partes de tu proyecto funcionan bien en todo momento.

En Butler, por ejemplo, existen varios test que se pueden ejecutar en todo momento y que se ejecutan cada vez que se compila el proyecto y ejecuta desde *gradle*.

Se ha utilizado *JUnit* para la realización de estas pruebas. [W18]

Algún ejemplo podría ser que el validador confirme que un fichero de configuración correctamente formado confirma que lo es y que se generen los ficheros necesarios, o que otros mal formados, devuelva el fallo del fallo concreto.

Para tener una versión estable, se han de hacer más comprobaciones y sobretodo más exhaustivas probando todas las casuísticas.

Para esto lo mejor es ir testeando poco a poco durante el desarrollo a través de test unitarios, tanto de caja negra como de caja blanca. Una vez se tenga el sistema final, se debería considerar una iteración de una semana probando el sistema a fondo con todas las posibilidades posibles. Es muy importante que la persona que lo haga no sea el desarrollador porque, inconscientemente, este va a hacer que el sistema falle lo menos posible.

En este proyecto se ha realizado sobre todo mayor inciso en este tema en Butler, dado que es el núcleo del proyecto. Y además se ha añadido mucho *feedback* frente al usuario cuando existan incongruencias o errores en los comandos que está realizando para avisar qué pasa, por qué y darle un consejo.

Ejemplo: intentar arrancar un contenedor cuya imagen no existe. Se avisa al usuario e la situación y se le sugiere crear la imagen previa.

## 7. Resultados

El resultado obtenido es un pequeño sistema básico en comparación con lo que podría llegar a ser en un futuro, dado que en el desarrollo incremental hay funcionalidades que se han dejado para posteriores iteraciones para poder llegar a todo lo que abarca el sistema y tener mínimo una versión funcional de todo el conjunto del sistema.

Como resultado tenemos un sistema que permite crear de forma personalizada, monitorizar crawlers y buscar sobre toda la información recolectada.

Hablando un poco en términos numéricos, se podría hablar de varios aspectos.

Coste de tiempo de realización de tareas sin contar el tiempo que tarda el usuario en meter los datos:

Creación de un proyecto: 1.5 segundos

Creación de una imagen de Docker con todo el sistema de crawling + OS: 2.5 segundos

Creación de un contenedor de Docker con el crawler preparado para ejecutarse: 1.5 segundos

Respecto al motor de búsqueda creado, la búsqueda de una query frente a todo el contenido es instantánea previa indexación.

Este tiempo se vería disminuido en 1.5 segundos, si Butler, en vez de ejecutarse desde un *jar* en el servidor, no fuera una aplicación *Spring Shell* y el código fuente estuviera en el servidor, así la ejecución sería instantánea, pero no se tendría la versión Shell.

Las pruebas se han realizado en un ordenador Intel i7. Procesador de 4 núcleos de 1.7 Ghz, tarjeta gráfica GTX 550.

La rapidez de ejecución del crawler ya depende de muchos factores: potencia del computador donde se esté ejecutando, dependiendo de las páginas que se visiten y su política frente a los crawlers (Robot.txt), si se han añadido plugins, depende de la eficiencia de estos.

Pero realmente lo valioso de este proyecto, es el sistema de Butler y Docker juntos. Su utilidad, estructura, modularidad y la potencia que tiene y sobre todo que puede tener con un poco de trabajo punto 8.1 Trabajo futuro.

Aquí se puede ver un ejemplo del resultado final del sistema:

<https://www.youtube.com/watch?v=L644A6WNCvI>

## 8. Conclusiones

En este proyecto se ha conseguido unificar la potencia y flexibilidad de Docker con un sistema que necesita de esta potencia, el cual no está nada explotado y que para según qué tipo de empresas especializadas en sistemas de la información puede ser clave para ahorrar muchísimo tiempo o no depender de empresas externas gracias a ser un proyecto de software libre.

El proceso costoso de crear crawlers personalizados, se ve solucionado a través de Butler, un sistema que permite, tanto vía web como vía línea de comandos, crear y personalizar tus propios crawlers y poder monitorizar su estado y con un motor de búsqueda propio sobre la información recolectada.

A través de un proceso sencillo con un DSL propio, el usuario puede crear casi instantáneamente sus crawlers incluso pudiéndoles inyectar código propio (los plugins), haciéndole posible poder obtener información que otros motores de búsquedas de otras empresas conocidas como *Google*, no llegan.

A pesar de haber obtenido un resultado más que gratificante, este proyecto, para poder tener un nivel competitivo y de utilidad real, necesita ser mejorado en algunos campos *punto 8.1 Trabajo futuro*.

La duración del proyecto y el reparto del tiempo en diferentes tareas aparece en el *Anexo F*.

### 8.1 Trabajo futuro

Se ha construido un sistema básico, pero la idea inicial y sobre la que se ha construido era un sistema más grande, que dado el peso de este trabajo no era viable realizar, pero sí que se ha logrado tener un sistema final potente y útil, y el cual, sería muy fácil de evolucionar a algo más grande tal como ha sido diseñado.

Las ideas que se quería haber podido tener tiempo para implementar en el sistema para dotarle de una gran capacidad son:

- Mayor configurabilidad en Butler. Aumentar el DSL para que soporte más opciones en la configuración.
- Aumentar el número de sistemas de crawling y S.O. que se pueden seleccionar, ahora mismo tan solo se puede elegir *Nutch* y *Ubuntu*.
- Cambiar la estructura del sistema a un sistema distribuido tal como se ha mencionado en 4. Tecnologías usadas, haciendo que los contenedores de Docker se ejecuten en máquinas externas.



- Integrar un sistema HDFS en el servidor central sustituyendo al FS actual.
- Integrar en el servidor central Solr (esto no sé si sería viable) una vez integrado el HDFS.
- Aumentar opcionalidades de monitorización, realizarlas más a tiempo real con sockets o colas de mensajes.
- Añadir nuevas funcionalidades como, por ejemplo, un recomendador de configuraciones, o un sistema de búsqueda general sobre toda la información recogida de todos (previa aceptación del usuario).

## 8.2 Valoración personal

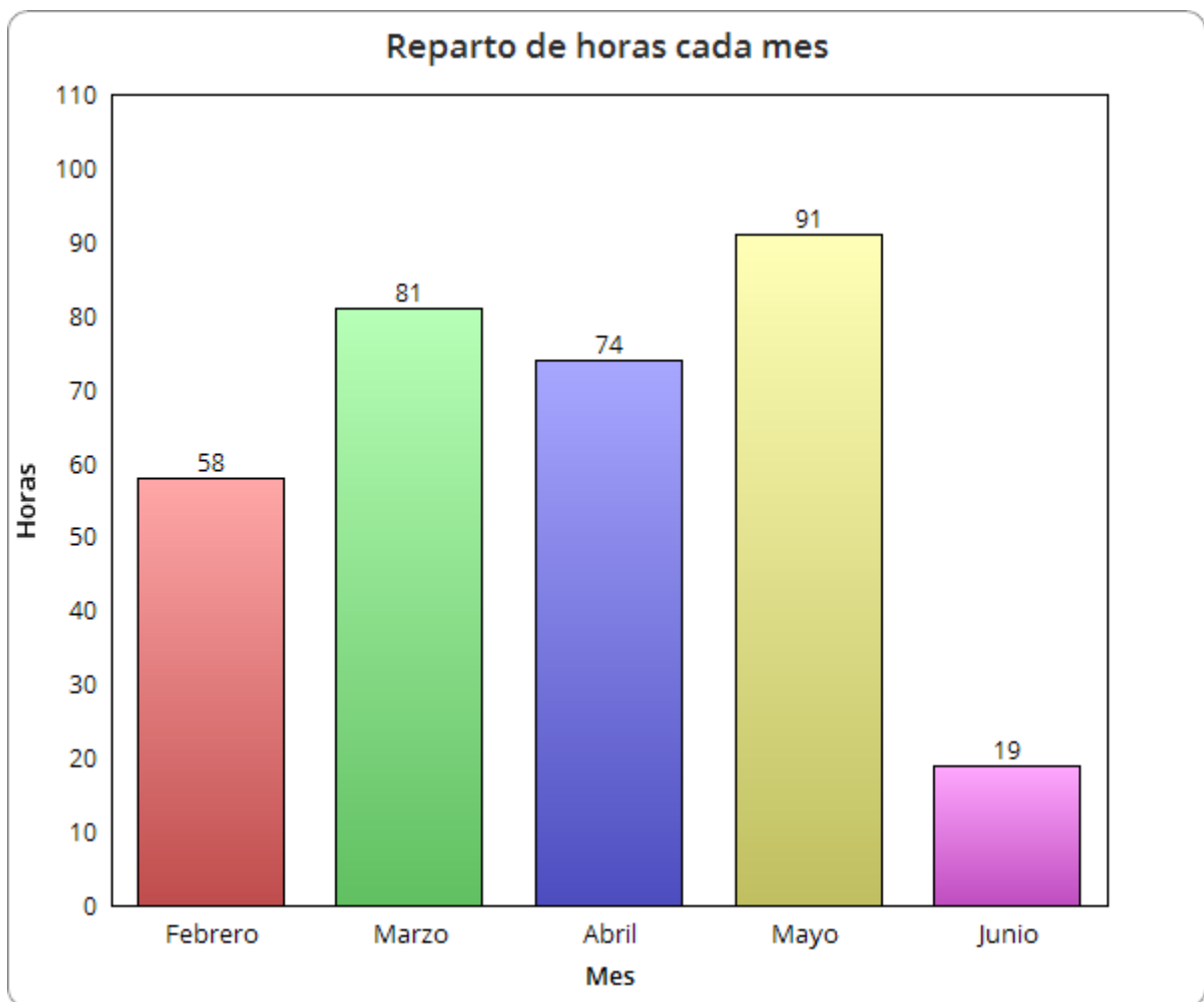
Una vez explicadas las conclusiones y el trabajo futuro, la valoración personal del proyecto, es totalmente positiva.

- Se han cumplido todos los objetivos que se propusieron en un principio sin excepción.
- He aprendido a realizar proyectos de mayor envergadura a los trabajos de universidad, utilizando un desarrollo incremental y el uso de pequeñas iteraciones en el desarrollo.
- He aprendido muchas tecnologías nuevas para mí y para las que no eran nuevas, he mejorado mi experiencia respecto a ellas, ya que era mínima porque no suelo hacer desarrollo web ni utilizar patrones de diseño.
- He mejorado mucho el hecho de enfrentarme a nuevas tecnologías y adaptarme más rápido a ellas.
- Sobre todo personalmente, he descubierto, respecto al desarrollo web, hacia donde van más mis preferencias, que era también uno de mis objetivos personales para este proyecto.
- El estudio previo sobre trabajos anteriores. Probar sobre sistemas ya implementados, trastear sobre ellos para intentar sacar el máximo de cada uno de ellos para aportar a mi proyecto, tampoco lo había hecho tan a fondo y la verdad es que se aprende mucho.

## 9. Trabajo y esfuerzo realizado y gestión del proyecto

El tiempo invertido en este proyecto asciende a 323 horas en total repartidas en varias tareas.

Para mostrar la evolución temporal del proyecto y la inversión del tiempo en este se van a mostrar imágenes que representen este hecho, ya que esta información se transmite mejor visualmente.



*Figura 2: Reparto de horas por mes del proyecto*

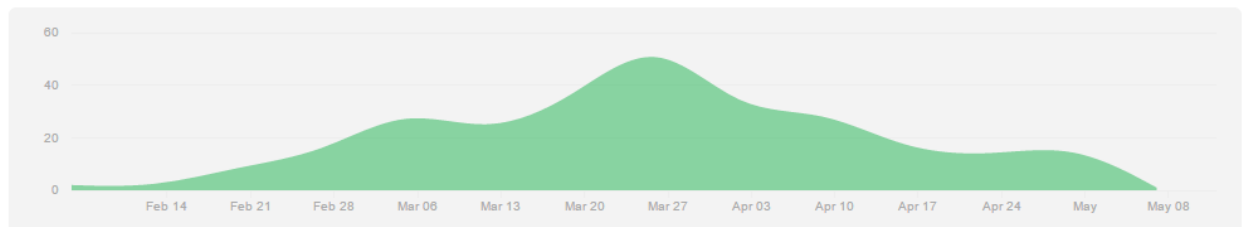


*Figura 3: Reparto de horas por tarea del proyecto*

Imágenes de GitHub referentes a los *commits* y a la evolución del código del proyecto.



*Figura 4: Número de commits realizados en el proyecto*



*Figura 5: Cantidad de código subido a GitHub durante el proyecto*

## Febrero

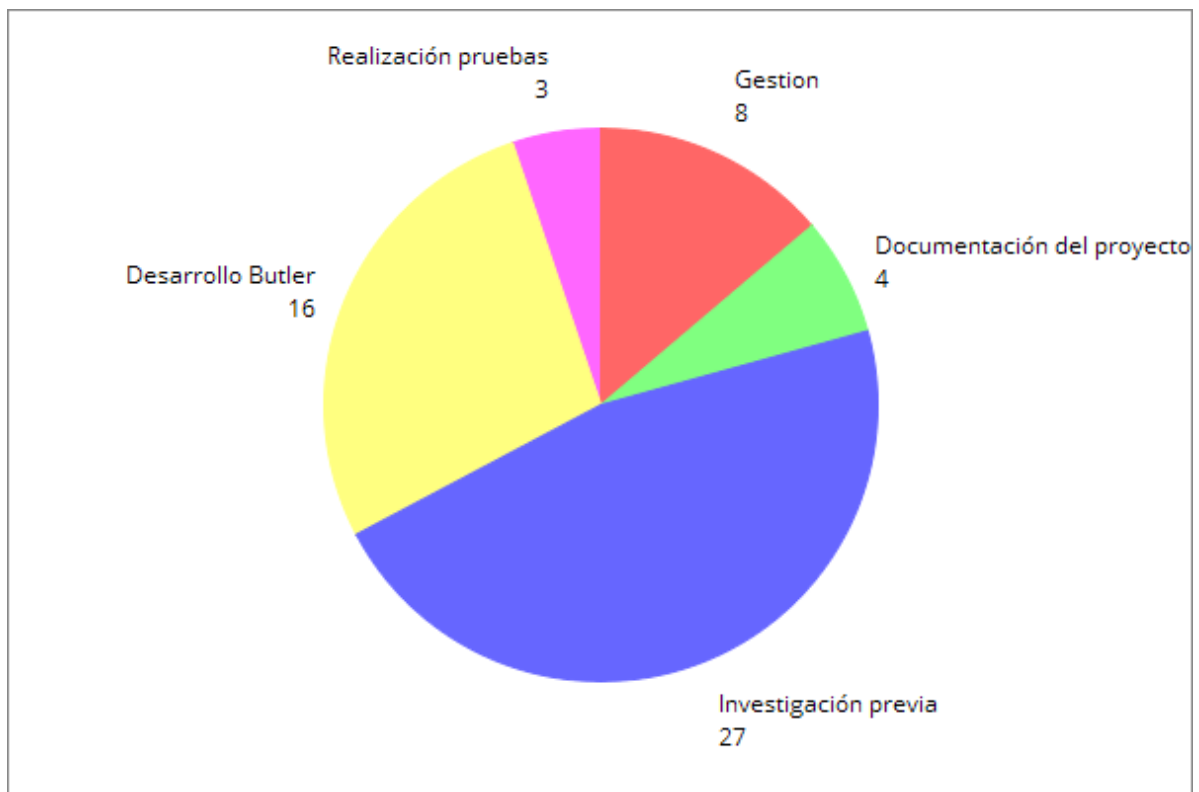


Figura 6: Reparto de horas en el mes de febrero por tareas

## Marzo

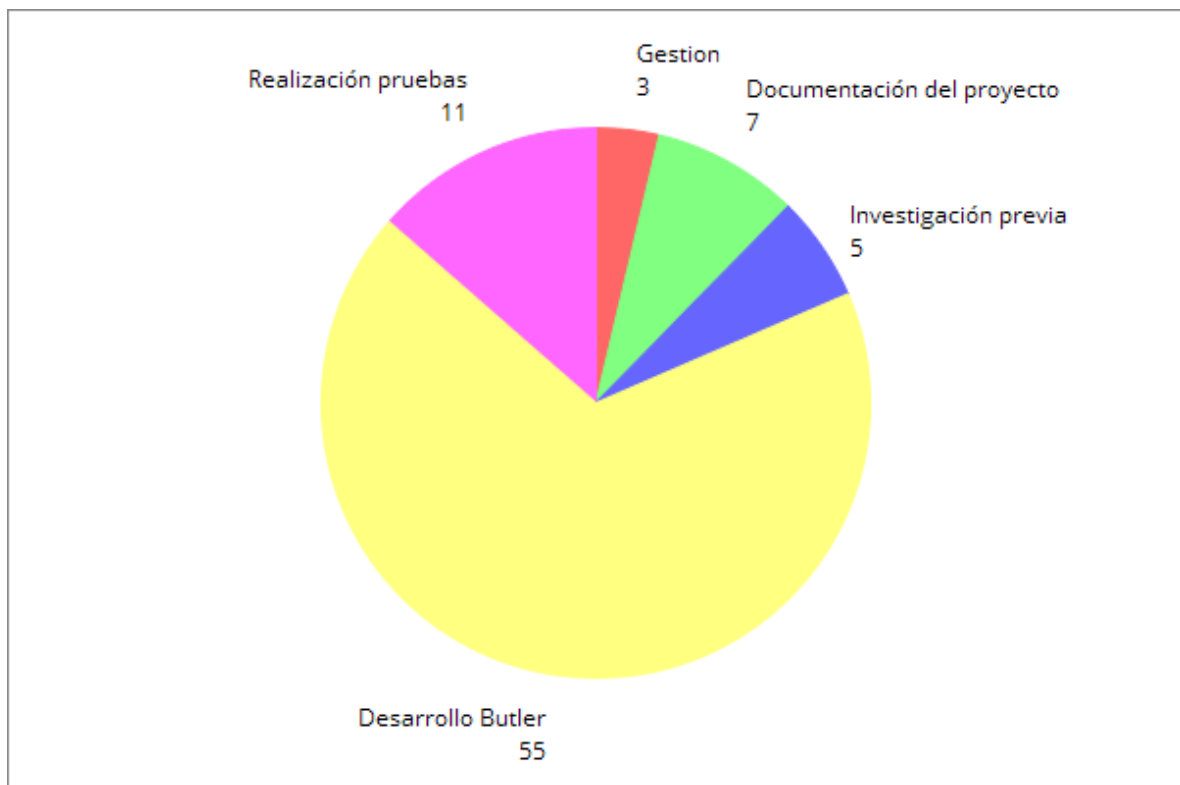


Figura 7: Reparto de horas en el mes de marzo por tarea

## Abril



Figura 8: Reparto de horas en el mes de abril por tareas

## Mayo

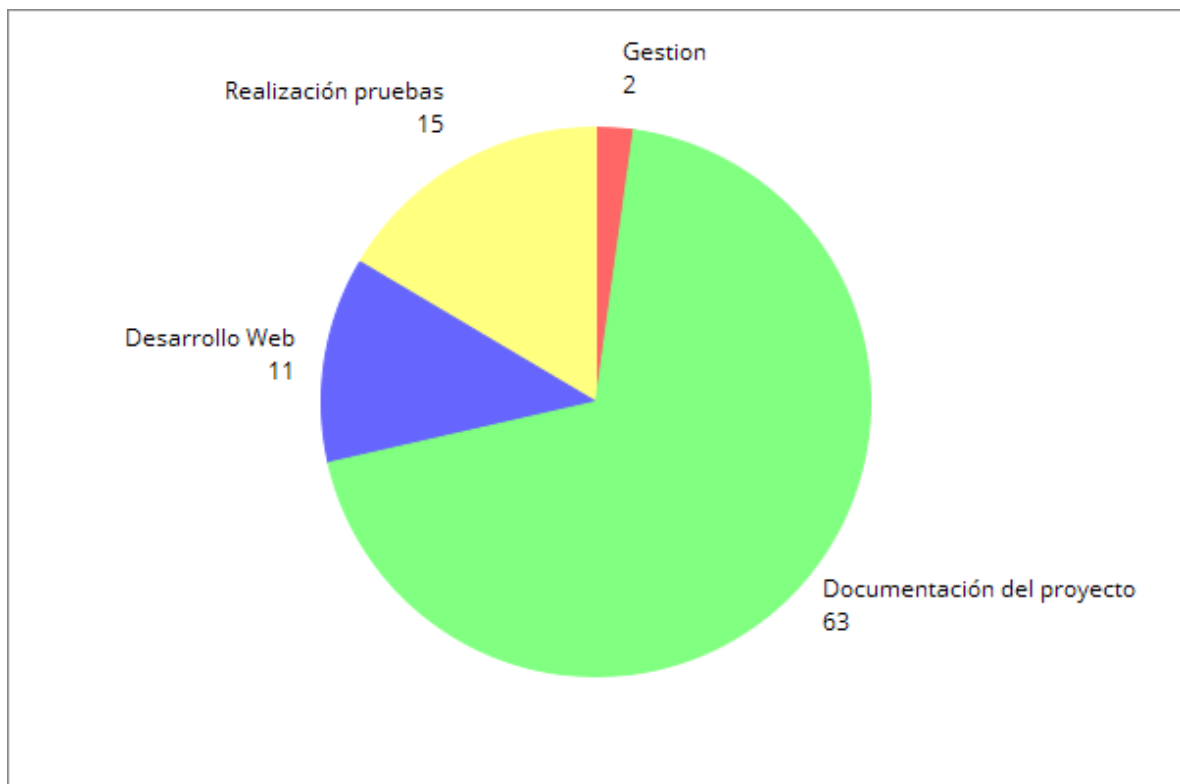


Figura 9: Reparto de horas en el mes de mayo por tareas

## Junio



*Figura 10: Reparto de horas en el mes de junio por tareas*

## 10. Bibliografía

Los libros o páginas webs que se han utilizado para documentar el Proyecto (son citados en este documento) o de guía, inspiración y aprendizaje para la construcción del proyecto son los siguientes:

[L1] Felix Bachmann, Len Bass, Paul C. Clements, David Garlan, James Ivers, Reed Little, Paulo Merson, Robert Nord, Judith A. Stafford. (2010). *Documenting Software Architectures: Views and Beyond, Second Edition*: Addison-Wesley Professional.

[L2] Artur Ejsmont. (2015). *Web Scalability for Startup Engineers*: Mc Graw Hill Education.

[W1] Docker (2016). *Docker docs*. Recuperado de <https://docs.docker.com>

[W2] Kirk Knoerdnschild (2015). *Patterns of modular architecture*. Recuperado de <https://dzone.com/refcardz/patterns-modular-architecture>

[W3] Íñigo Alonso (2015). *Docker desplegando aplicaciones*. Recuperado de <http://101breakpoints.com/docker-desplegando-aplicaciones>

[W4] Spring (2014). *Spring Shell documentation*. Recuperado de <http://docs.spring.io/spring-shell/docs/current/reference/htmlsingle>  
<http://docs.spring.io/autorepo/docs/spring-shell/1.2.0.M1/reference/html/dev-shell.html>

[W5] Google (2016). *AngularJs documentation*. Recuperado de <https://docs.angularjs.org/api>

[W6] Apache (2015). *Nutch documentation*. Recuperado de <http://nutch.apache.org/apidocs/apidocs-1.9/index.html>  
<https://wiki.apache.org/nutch/NutchTutorial>

[W7] Spring (2016). *Spring Guides*. Recuperado de <https://spring.io/guides>

[W8] Spring (2015). *Creating and using Bean definitions*. Recuperado de <http://docs.spring.io/spring-javaconfig/docs/1.0.0.m3/reference/html/creating-bean-definitions.html>

[W9] Apache (2015). *DockerFile for Nutch 2.x*. Recuperado de <https://issues.apache.org/jira/browse/NUTCH-190>

[W10] Apache (2014). *The Hadoop Distributed File System*. Recuperado de <https://developer.yahoo.com/hadoop/tutorial/module2.html>

[W11] Docker (2015). *Docker hub*. Recuperado de <https://hub.docker.com>

[W12] Github (2016), *Github*. Recuperado de <https://github.com>

[W13] Nathan Sweet (2016), *Yalm for java*. Recuperado de <http://yamlbeans.sourceforge.net>

[W14] David Geary (2003), *simply singleton*. Recuperado de <http://www.javaworld.com/article/2073352/core-java/simply-singleton.html>

[W15] Cecilio Álvarez (2014), *Ejemplo de java singleton: Patrones classloaders*. Recuperado de <http://www.arquitecturajava.com/ejemplo-de-java-singleton-patrones-classloaders>

[W16] James Sugrue (2010), *Adapter pattern tutorial with java*. Recuperado de <https://dzone.com/articles/design-patterns-uncovered-0>

[W17] Apache (2016), *Lucene documentation*. Recuperado de <https://lucene.apache.org/core/documentation.html>

[W18] Alvin Reyes (2016), *Junit integration test example*. Recuperado de <https://examples.javacodegeeks.com/core-java/junit/junit-integration-test-example>

[W19] Quora (2016). *What are the best web crawling services?* Recuperado de <https://www.quora.com/What-are-the-best-web-crawling-services>



## Anexos

### Anexo A. Butler en detalle. Guías y esquemas.

Butler está compuesto de tres grandes partes.

- El validador de ficheros de configuración
- El constructor de los ficheros finales
- Comandos para el uso de estos ficheros y manejo de los sistemas de crawling.

Dejando los comandos aparte, ya que son mayoritariamente operaciones de Docker y *Nutch* encapsuladas junto con lógica de negocio, la parte del validador y el constructor, se puede apreciar muy bien con un diagrama de actividad.

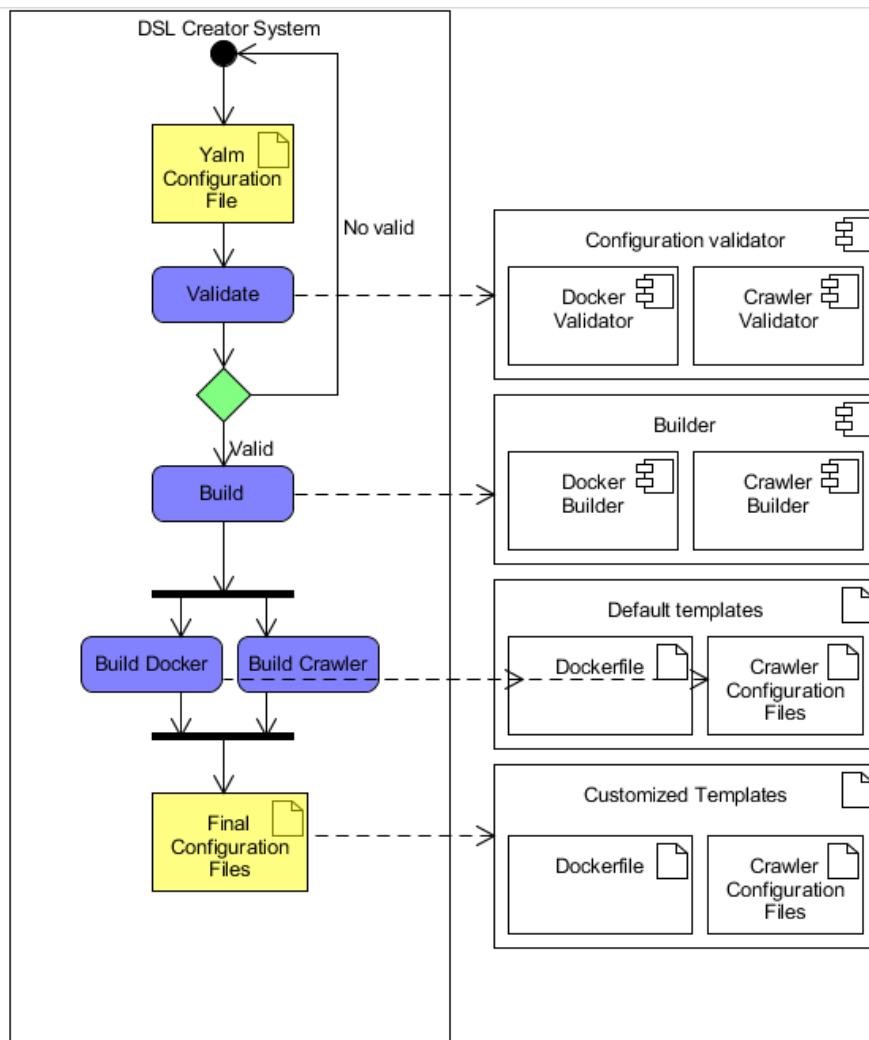


Figura 11: Diagrama de actividad del validador y constructor de Butler.

Se puede apreciar perfectamente cuál es el funcionamiento del sistema y a qué elementos del sistema está asociado cada paso.

1. Dado un fichero de configuración de tipo yalm, el sistema valida la construcción de este según a la especificación del DSL que se verá en el siguiente punto del anexo. Estos validadores son los validadores que se han nombrado en 3.1.2 Validador.
2. Si el fichero es válido, el siguiente paso es la construcción de los ficheros de configuración de salida necesarios para el sistema de crawling a realizar, también existen dos módulos, uno se encarga de la parte de Docker y otro de la parte del Crawler (con un patrón Adapter para diferentes sistemas de crawling).
3. Estos módulos utilizan templates por defecto integrados en el sistema a partir de los cuales se formarán los ficheros de salida finales.
4. El módulo que construye los ficheros de Docker necesarios, forma el Dockerfile, y el módulo del crawler forma los ficheros de configuración necesarios para realizar las configuraciones especificadas en el DSL respecto al sistema de crawling seleccionado.

El validador, como ya se ha explicado en 3.1.2 Validador, tiene un validador para cada característica de configuración. Cada uno de ellos implementa un interfaz validator para estandarizar los métodos a implementar y una mayor organización.

Se puede configurar el validador de Butler a través de las *Beans* de Java. Ahora mismo se utiliza un solo validador que está formado por dos validadores, uno que valida la parte de Docker y otro que valida la parte del sistema crawling (y que ambos están formados por más validadores).

## Especificación del DSL

Para crear el sistema de crawling se ha de especificar ciertos aspectos del sistema.

En lo que se refiere al contenedor Docker, se ha de especificar el sistema operativo sobre el que va a correr el sistema de crawling. Del OS se debe especificar el nombre y la versión de esta forma:

*#Docker container Operating System*

*dockerOS:*

*name: ubuntu*

*version: 14.04*

Respecto al sistema de crawling, se deben especificar aspectos sobre qué sistema es y sobre su configuración que se aplicará en el momento en el que este se empiece a ejecutar.

- Configuraciones mínimas obligatorias
  - El nombre y la versión del sistema de crawling.
  - Las semillas que servirán como base al sistema.
  - El número de rondas o iteraciones que el sistema permanecerá en ejecución.
  - Si la extracción de información del crawler se debe realizar en cada ronda o tan solo al terminar su ejecución.
  - El tipo de información a extraer.

Especificación a nivel de fichero:

*dockerOS:*

*name: Es el nombre del sistema de operativo, debe ser compatible con el sistema de crawling elegido. Hasta ahora tan solo se permite algunos sistemas Linux como Ubuntu*

*versión: Versión del sistema de operativo, se puede elegir cualquiera disponible.*

*crawlSystem:*

*name: Es el nombre del sistema de crawling, hasta ahora tan solo se puede elegir nutch.*

*versión: Versión del sistema de crawling, se puede elegir cualquiera disponible.*

*Seeds: Semillas url para el crawler. Cada una se separa por un salto de línea previo ‘-’.*

*Rounds: Número de iteraciones que va a realizar el crawler, el nivel de profundidad hasta el cual va a llegar.*

*Extraction: Si la indexación interna va a ser en cada iteración o únicamente a la finalización del crawler. Posibles valores [rounds | finish]*

*infoCrawled: El formato de la información que le crawler va a extraer. Posibles valores [text | html].*

*queueMode: Determina la organización de colas del crawler. Por defecto tendrá el valor ‘byHost’. Posibles valores [byHost | byDomain | byIP]*

- Configuraciones opcionales
  - Tipo o modo de cola.
  - Información respecto a timeouts o tiempos de delay.
  - Restricciones sobre la información a recoger.

**crawlSystem:**

*maxLength: Longitud máxima de los ficheros a descargar. Valor por defecto 65536 bytes.*

*maxCrawlDelay: el tiempo máximo que el crawler espera para hacer fetch a la url siguiente. Si se toma un valor menor al permitido por la página por el fichero robot.txt, la url no se fetcheará. Si el valor es ‘-1’, nunca se saltará ninguna página y esperará lo que diga el robot.txt*

**Timeouts:**

*Parser: timeout en segundos máximo para parsear un documento.*

*fetchTimes: el número de veces que el crawler reintentará hacer el fetch de una url que falle.*

*network: timeout de la red en milisegundos máximo a esperar.*

Los plugins no serán especificados en el fichero de configuración. Estos irán en una carpeta llamada 'plugins' al mismo nivel que el fichero de configuración realizado. Esta carpeta debe tener dentro, una carpeta por cada plugin que se quiera añadir. Dichas carpetas tendrán el nombre del plugin y su contenido deberá ser los ficheros jars del mismo y el fichero plugin.xml con su especificación. Estos plugins deben ser plugins pre-compilados.

En el caso de que haya algún problema en la validación de cualquier fichero de configuración, se indicará el primer error encontrado especificando el tipo de error.

Un ejemplo simple de lo que sería un fichero de configuración sería este:

*dockerOS:*

*name: ubuntu*  
*version: 14.04*

*crawlSystem:*

*name: nutch*  
*version: 1.9*  
*seeds:*  
*- https://eina.unizar.es/*  
*- http://www.unizar.es/*  
*rounds: 2*  
*extraction: round*  
*infoCrawled: text*  
*queueMode: byHost*

## Guía de usuario

Requisitos para instalar y ejecutar Butler:

- Tener instalado Java 8
- Tener instalado Docker y en ejecución. En un apartado de los Anexos más adelante se mostrará cómo realizar la instalación.

Una vez se cumplan estos requisitos, para descargar Butler tan solo hay que descargárselo del repositorio de GitHub <https://github.com/Shathe/101CrawlersWeb>

La descarga se puede hacer tanto clonando el repositorio desde línea de comandos o aplicación de Git o desde la página web descargándolo como *Zip*.

Para ejecutarlo, tan solo hay que ejecutar el script que está en el nivel superior del proyecto llamado `butler.sh`, el cual compila el proyecto, realiza los test que asegura que el proyecto funciona y ejecuta el sistema.

Utilizando el fichero de configuración de ejemplo del apartado anterior, se va a mostrar como sería una ejecución de todos los comandos en su orden natural.

Para no complicar a los usuarios en su primera toma de contacto con Butler, en este proyecto ya hay un fichero de configuración creado (el mismo con el que se va a realizar el ejemplo), llamado `conf_tutorial.yml`.

Una vez ejecutemos el script `butler.sh` se nos abrirá en la consola Butler y nos permitirá ver los comandos disponibles a realizar a través del tabulador:

```
=====
*                                     *
*           101Crawlers               *
*                                     *
=====
Version:0.0.1
Welcome to 101Crawlers CLI
101shell>config --file conf_tutorial.yml --idProject 5
Configured successfully
101shell>build --idProject 5 --imageName nueva
```

*Figura 12: Inicialización de Butler.*

Ahora vamos a realizar paso a paso, todos los comandos que ofrece Butler:

### 1. Ejecutar la configuración:

```
config --file conf_tutorial.yml --idProject 5
```

#### Salida esperada:

```
configured successfully
```

En este paso el sistema validará el fichero de configuración. En el caso de que exista algún error indicará donde y cual. En caso de que todo sea correcto, creará los ficheros necesarios.

### 2. Ejecutar su construcción:

```
build --idProject 5 --imageName nueva
```

#### Salida esperada:

```
Image built successfully
```

En este paso, el sistema creará la imagen de Docker especificada por el usuario con el sistema de crawling y sus configuraciones que se hayan especificado.

### 3. Ejecutar el arranque de docker:

```
start --containerName container --idProject 5 --imageName nueva
```

#### Salida esperada:

```
Container started
```

Ahora mismo el contenedor de Docker está en marcha. Solo está en marcha el sistema operativo en el contenedor, el siguiente paso es poner el crawler en marcha. En caso de que el contenedor haya sido pausado o parado, este comando servirá también para reanudarlo.

### 4. Ejecutar el arranque del crawler:

```
run --containerName container --idProject 5 --imageName nueva
```

#### Salida esperada:

```
Crawler started
```

Ahora mismo el crawler está en marcha y se ejecutará las rondas que haya sido configurado o hasta que ya no tenga enlaces. Ahora ya solo hay que esperar a recopilar la información.

5. Ahora tengo que esperar (o no) a que termine el crawler de ejecutarse, ¿Cómo lo puedo saber?

Vamos a ejecutar un comando para saber información sobre el crawler

```
info --containerName container --idProject 5 --imageName nueva
```

Salida esperada:

```
Running
```

o el estado del contenedor

```
status --containerName container --idProject 5 --imageName nueva
```

Salida esperada:

```
Running
```

Ahora vamos a esperar unos minutos a que termine [5/10 minutos después] y ejecutamos:

```
finished --containerName container --idProject 5 --imageName nueva
```

Las dos posibles salidas son:

```
The crawler hasn't finished yet
```

Si no ha acabado, como nada más ejecutarlo. O, por el contrario, si en mi caso, he esperado unos diez minutos

Yes, the crawler has finished

Si quieres saber algo más sobre la información del crawler, también puedes probar con esto:

```
runningStatus --containerName container --idProject 5 --imageName nueva
```

Salida esperada:

```
Fetches links: 2, unfetched links: 170, rounds: 1/2
```

Esto significa que el crawler lleva 1 de 2 rondas realizadas, que ha recopilado información de 2 links y que, en la siguiente ronda, valorará extraer información de 170 links.



## 6.Extraer la información:

Lo primero de todo, hay que tener en cuenta que antes de poder buscar, hay que indexar la información. Como hemos elegido "extraction: round" se irá actualizando el índice solo, pero, de todas formas, podemos hacerlo manualmente.

```
index --containerName container --idProject 5 --imageName nueva
```

### Salida esperada:

```
Indexed correctly
```

Y ahora para realizar una búsqueda tan solo tenemos que hacer esto:

```
search --query 'unizar universidad' --containerName container --idProject 5 --imageName nueva --top 5
```

### Salida esperada:

```
58 total matching documents
http://paper.li/CatedrasUnizar/1361728396
https://twitter.com/unizar
https://twitter.com/EINAunizar
http://paper.li/OTRI_Unizar/1374046234
http://www.unizar.es/
Results shown
```

Los resultados están ordenados de mayor a menos importancia respecto a nuestra búsqueda. Podemos especificar incluso el máximo de resultados que queremos ver en la consola. Si no lo hacemos, nos devolverá todos los resultados. Además, el sistema creará un documento con todos los resultados.

### Comandos útiles pero prescindibles:

Si en algún momento quiero parar la ejecución del crawler tan solo tengo que ejecutar:

```
stopCrawl --containerName container --idProject 5 --imageName nueva
```

### Salida esperada:

```
Crawl stopped correctly
```

Si quiero parar el contenedor porque, por ejemplo, quiero pararlo para eliminarlo, tan solo tengo que ejecutar:

```
stopContainer --containerName container --idProject 5 --imageName nueva
```

### Salida esperada:

```
Container stopped correctly
```

Esto sirve para poder eliminarlo, cuando el crawler haya terminado y ya no se necesite gastar recursos de procesador y memoria RAM.

Si quiero pausar el contenedor porque, por ejemplo, no quiero que avance de momento o no quiero gastar recursos, ejecuto:

```
pauseContainer --containerName container --idProject 5 --
imageName nueva
```

Salida esperada:

```
Container paused correctly
```

Si quiero borrar el contenedor tan solo tengo que ejecutar:

```
deleteContainer --containerName container --idProject 5 --
imageName nueva
```

Salida esperada:

```
Container deleted correctly
```

Y si en cambio lo que quiero borrar es la imagen tan solo tengo que ejecutar:

```
deleteImage --idProject 5 --imageName nueva
```

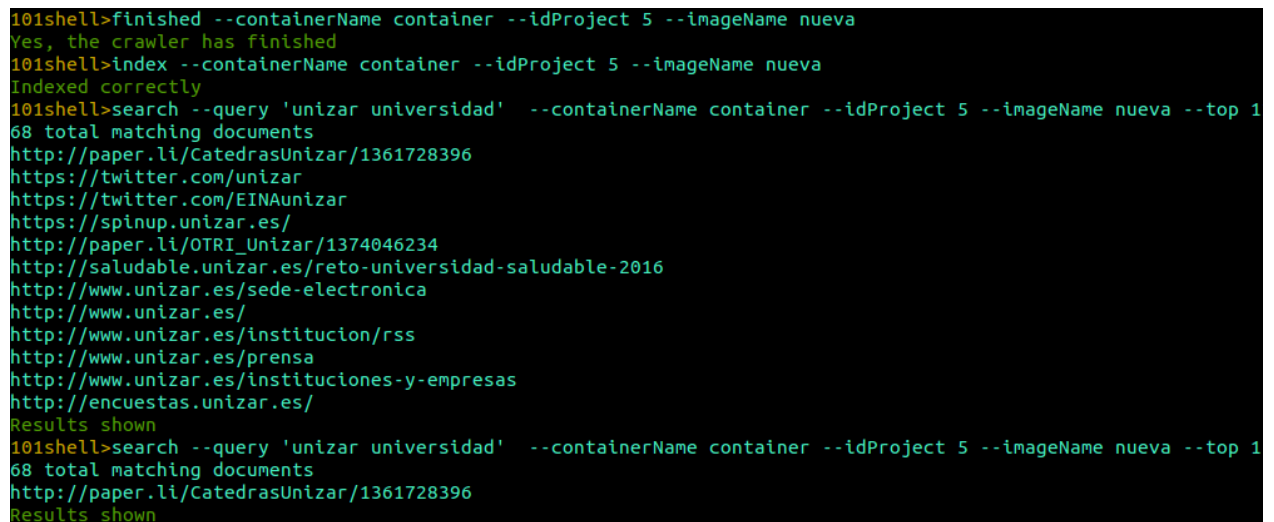
Salida esperada:

```
Image deleted correctly
```

Los comandos de borrar suelen ser útiles cuando el crawler ha terminado y se quiere de dejar usar recursos de disco (almacenamiento).

Si existe algún error al ejecutar los comandos, se avisará por pantalla de qué error es.

Visualmente, algunas de estas ejecuciones quedarían así:



```
101shell>finished --containerName container --idProject 5 --imageName nueva
Yes, the crawler has finished
101shell>index --containerName container --idProject 5 --imageName nueva
Indexed correctly
101shell>search --query 'unizar universidad' --containerName container --idProject 5 --imageName nueva --top 1
68 total matching documents
http://paper.li/CatedrasUnizar/1361728396
https://twitter.com/unizar
https://twitter.com/EINAunizar
https://spinup.unizar.es/
http://paper.li/OTRI_Unizar/1374046234
http://saludable.unizar.es/reto-universidad-saludable-2016
http://www.unizar.es/sede-electronica
http://www.unizar.es/
http://www.unizar.es/institucion/rss
http://www.unizar.es/prensa
http://www.unizar.es/instituciones-y-empresas
http://encuestas.unizar.es/
Results shown
101shell>search --query 'unizar universidad' --containerName container --idProject 5 --imageName nueva --top 1
68 total matching documents
http://paper.li/CatedrasUnizar/1361728396
Results shown
```

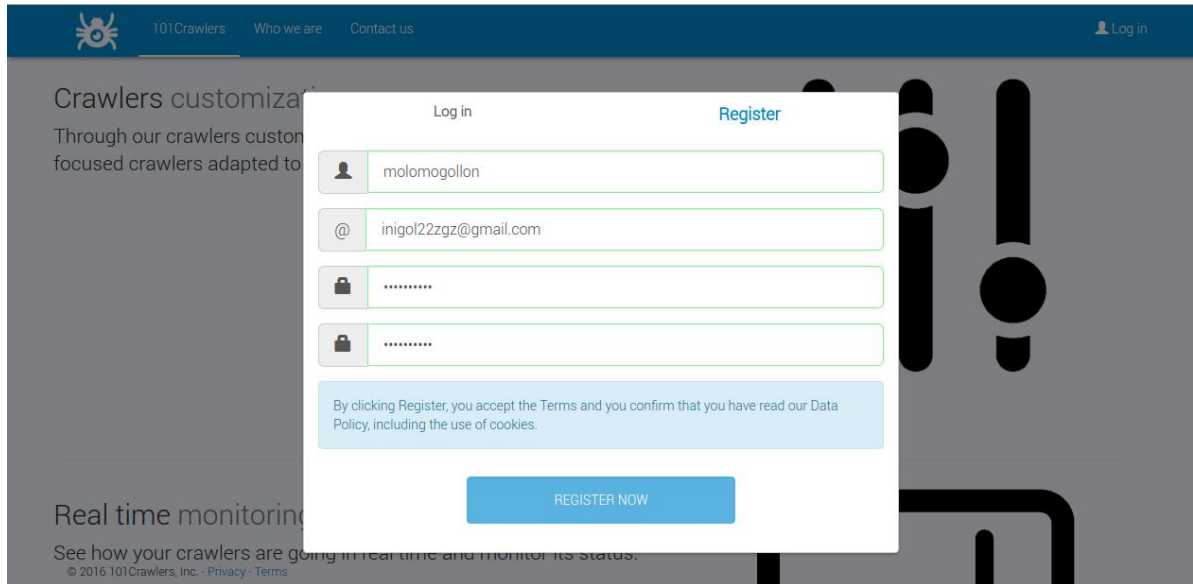
*Figura 13: Ejecución de comandos de Butler.*

Donde se puede apreciar los comandos ejecutados, como la indexación y búsqueda de algunas queries, una vez sabemos que el crawler ha terminado su ejecución.

## Anexo B. Sistema web

### Guía de usuario

Para utilizar el sistema web, se ha de crear un usuario propio para poder acceder tan solo a los crawlers creados por cada persona.

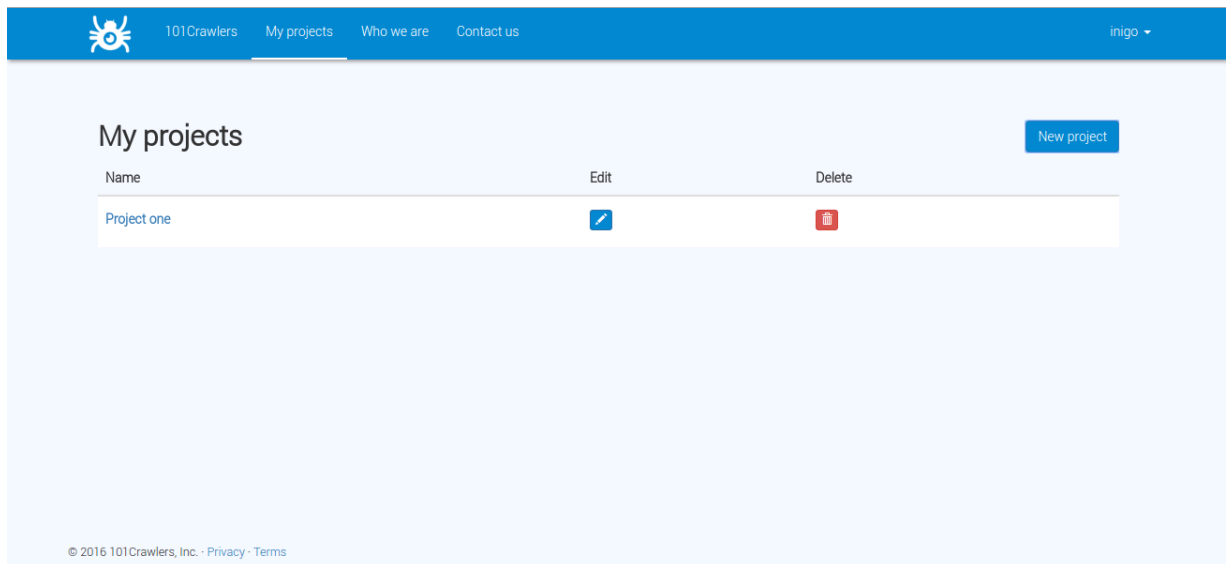
The image shows a web browser window with the 101Crawlers website. A modal form is displayed in the center for user registration. The form has a dark blue header with the 101Crawlers logo and navigation links. The main content area is white with a light blue background. It contains a 'Log in' link and a 'Register' link. Below these are four input fields: a username field (containing 'molomogollon'), an email field (containing 'inigol22zg@gmail.com'), and two password fields (both containing '\*\*\*\*\*'). A blue box contains the text: 'By clicking Register, you accept the Terms and you confirm that you have read our Data Policy, including the use of cookies.' At the bottom is a blue button labeled 'REGISTER NOW'.

*Figura 14: Registro de un usuario.*

Una vez se ha creado un usuario y se ha iniciado sesión (Log in), es momento de crear un proyecto, para hacer esto, hay que darle al botón de 'new project' y se mostrará una nueva pantalla para rellenar los datos necesarios para la creación del proyecto.

Información mínima necesaria:

- Nombre del proyecto
- Fichero de configuración que siga la especificación explicada en el anexo *especificación del DSL*



*Figura 15: Creación de un proyecto.*

Tras introducir el nombre y seleccionar el fichero correspondiente al DSL, el fichero se tiene que subir al servidor, así que, para poder subir el fichero de configuración, se ha de pulsar a 'upload DSL file'

Una vez subido el fichero, se puede dar a guardar el proyecto. Cuando se le dé, se validará el fichero y si es correcto, el proyecto se creará, sino es así, se notificará del error y no se creará.

Información opcional:

- Plugins

Los plugins se deben de adjuntar con todos sus ficheros de golpe y ya compilados, es decir, el fichero plugin.xml y los jars. Para poder añadir un plugin, tan solo hay que darle un nombre (el nombre del plugin que viene en el plugin.xml) y adjuntar los ficheros. Al darle a Upload plugin, se subirá el plugin al servidor.

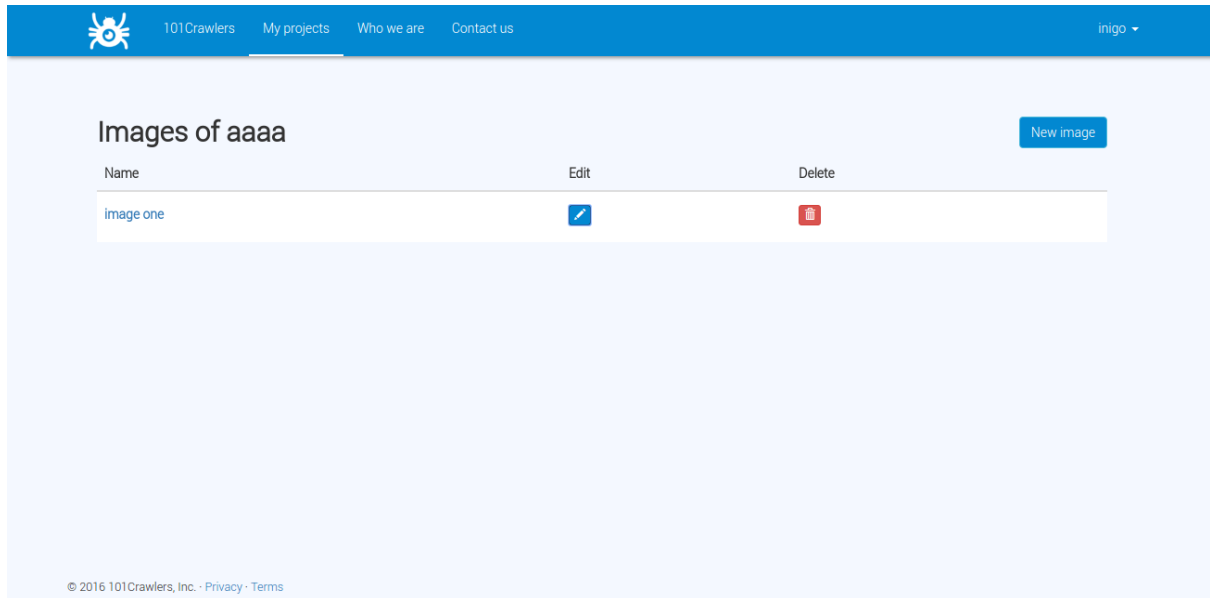
Se pueden añadir un número ilimitado de plugins, pero sus ficheros como máximo pueden ocupar 10 megas.

Si en cualquier momento se quiere resetear las configuraciones/plugins subidos, tan solo se debe clicar el botón 'reset upload'.

Una vez creado el proyecto, se puede editar el proyecto, borrar el proyecto, o entrar dentro para crear imágenes y contenedores.

Una vez creado el proyecto, hay que clicarlo para poder manejarlo. El siguiente paso es la creación de una imagen del proyecto actualmente (una imagen de Docker con la configuración actual), ya que se puede editar la configuración del proyecto y poder tener imágenes en un mismo proyecto con diferente configuración.

Lo primero de todo, una imagen, es una forma de organización del proyecto. Dada la última configuración del proyecto, se toma una imagen de esta configuración, para poder crear instancias (contenedores) más adelante, y así poder tener en un mismo proyecto, diferentes configuraciones, guardadas en diferentes imágenes.



*Figura 16: Creación de una imagen del proyecto 'aaaa'.*

Al crear una imagen tan solo se le tiene que especificar el nombre. El proceso de creación de esta imagen puede tardar desde un segundo hasta varios minutos. Esto es debido a que crear una imagen es equivalente a Descargar el sistema operativo, configurarlo, descargar todo el software necesario para el sistema de crawling.

Puede llegar a tardar muy poco debido a la caché de Docker. Todo lo que se ha creado alguna vez para alguna imagen, se reutilizará para las demás, es decir, si yo creo una imagen con la misma configuración que otra que ya ha sido creada, tardará tan uno segundos en tener montado todo el sistema de la imagen.

También se pueden borrar o editar.

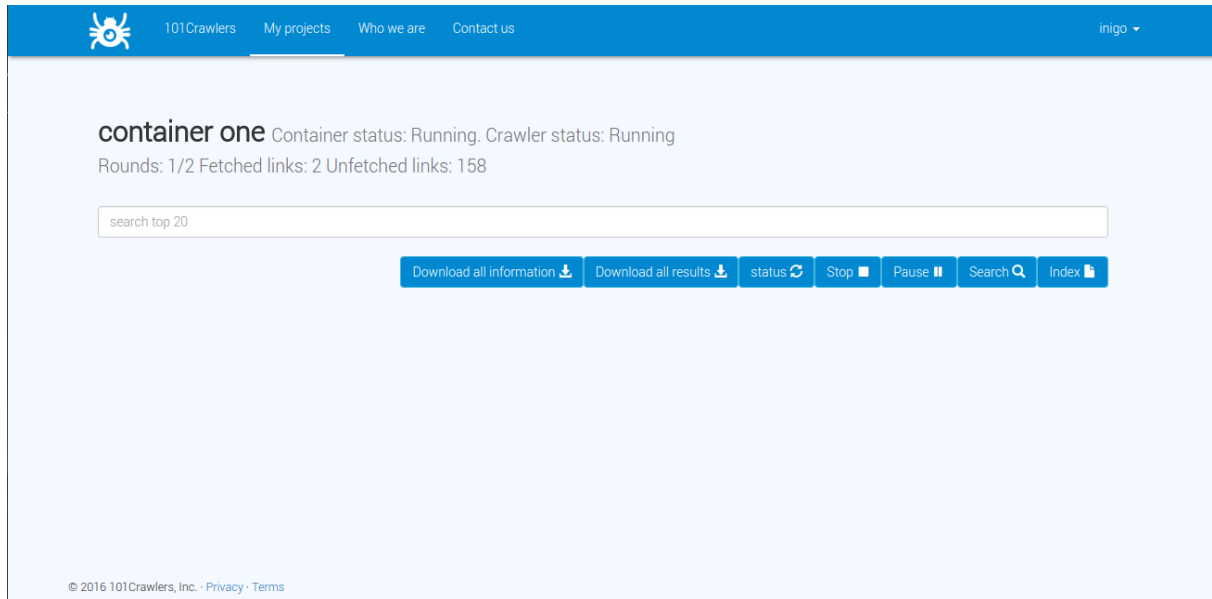
Para cada imagen, se crean contenedores Docker, para poder ejecutar esa imagen en el momento que se quiera y las veces que se quiera.

Los contenedores son instancias de imágenes, teniendo un contenedor, es como tener una máquina virtual, un ordenador funcionando teniendo el sistema de crawling dentro.

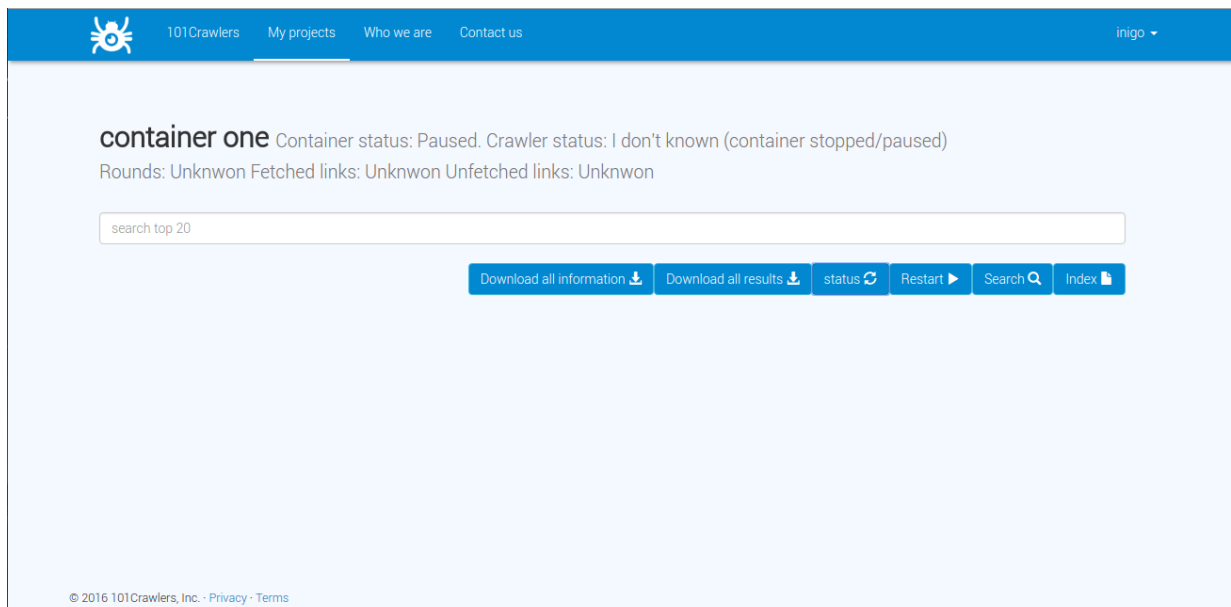
Para crearlo, previamente se ha debido de introducirse en una imagen. así tan solo hay que especificar el nombre tras darle a crear nuevo proyecto.

También se pueden borrar o editar.

Una vez lo creas, y clicas en el para introducirte dentro, tardará en cargar un segundo o dos el estado actual del contenedor y mostrará esta pantalla con múltiples funcionalidades.



*Figura 17: Contenedor 'one' ejecutándose.*



*Figura 18: Contenedor 'one' en estado pausado.*

Se pueden apreciar en ambas imágenes varias posibles operaciones según su estado:

- Index: Indexa todo lo que se ha ido recogiendo de información, esta opción está introducida debido a que en el DSL se especifica cuando se quiere indexar, pero se da la opción de realizarlo manualmente por si el usuario quiere asegurarse o ha cambiado de opinión respecto a cuándo realizarla
- Search: Puedes buscar cualquier contenido en la información que el sistema ha recogido tan solo introduciendo la búsqueda y clicando a este botón. Tan solo muestra los 20 primeros resultados
- Pause/Stop: Pausa o para el sistema para no consumir recursos. Se recomienda tan solo pausarlo
- Restart: Solo visible si se ha pausado o parado previamente, vuelve a iniciar en el estado anterior el sistema
- Download all results: Descarga en un fichero con TODOS los resultados de la query
- Download all: Descarga TODA la información recogida por el crawler

En la pantalla del contenedor, se muestra el estado del mismo en todo momento, así como el estado del crawler.

La diferencia entre ellos es que el contenedor, es como el ordenador, donde se está ejecutando el crawler, en cambio, el crawler es el programa que recoge toda la información.

En esta dirección web <https://www.youtube.com/watch?v=L644A6WNCvI> reside un ejemplo de uso.

## Anexo C. Validación

La validación del sistema es un elemento imprescindible.

Se ha comentado en 6. Validación y pruebas del sistema, las validaciones realizadas en este sistema, pero en este Anexo se va a centrar la atención a los test (con *JUnit*) de caja negra, y al desarrollo realizado por TDD.

En la versión actual del software, hay tres elementos que se están controlando a través de los test en Butler.

- Funcionalidades relacionadas con los comandos
- La correcta indexación del motor de búsqueda
- El correcto funcionamiento del Shell

Todo este código está organizado en un paquete llamado *test* para mantener el código organizado según sus funcionalidades.

Para la realización de este tipo de tests se han utilizado características de *java* y *Spring* para facilitar tanto el desarrollo como la configuración de estos.

- Beans. Objetos que se auto-instancian dada una configuración/instanciación por defecto previa. Se puede especificar para estas configuraciones los aspectos que se deben dar para que en algunos casos se instancien unas configuraciones y en otras ocasiones otras.

Así, por ejemplo, con una previa anotación java de *@autowired* se instancian validadores los ficheros de configuración que se necesiten para los tests. [W8]

- Anotaciones de java. Como las anotaciones de *@ActiveProfile*, *@Value* para que dependiendo del perfil de ejecución algunas variables tomen unos valores u otros, así se pueden tener variables de configuración como rutas, por ejemplo. También para definir con que se va a ejecutar los tests, definir que un método es un test etc...

Un ejemplo del código sería este:



```

/**
 * Test the configuration builder and validation
 */
@RunWith(SpringJUnit4ClassRunner.class)
@SpringApplicationConfiguration(Application.class)
@ActiveProfiles("test")
public class CoordinatorTest {

    @Autowired
    private ApplicationContext ctx;

    @Autowired
    private ConfigurationValidator configurationValidator;

    @Autowired
    private CrawlValidator crawlValidator;

    @Value("${butler.base}/")
    private String baseDir;

    @Autowired
    public Operations operations;

    /** Detects if a well formed configuration file, pass the validation */
    @Test
    public void detectEverythingIsOK() throws URISyntaxException {
        CrawlConfiguration config;
        config = readConfiguration("conf.yml");
        assertNotNull("YamlConfigRunner debe devolver una configuración y no null", config);
        System.out.println(config.toString());

        ValidationResult result = configurationValidator.validate(config);
        assertTrue("DefaultValidator debe devolver que está bien", result.isOk());
        assertEquals("DefaultValidator debe dar OK", Validator.Status.OK, result.getFirstErrorCode());
    }
}

```

*Figura 19: Ejemplo de test de Butler*

## Anexo D. Indexación

El proceso de indexación es muy necesario para que un motor de búsqueda sea eficiente. Butler utiliza Lucene como herramienta para realizar este proceso.

A grandes rasgos el proceso se ha explicado en 3.1.5 Indexador y buscador. Se ha hablado de la coordinación entre el contenedor Docker y Butler para mantener la uniformidad y consistencia de la información ya que existe un proceso automatizado que indexa dentro del contenedor Docker y debe utilizar el mismo código metodológica que en Butler, que es el otro elemento donde se puede dar esta indexación a petición del usuario.

Para aclarar el proceso, se muestra a continuación un diagrama de secuencia.

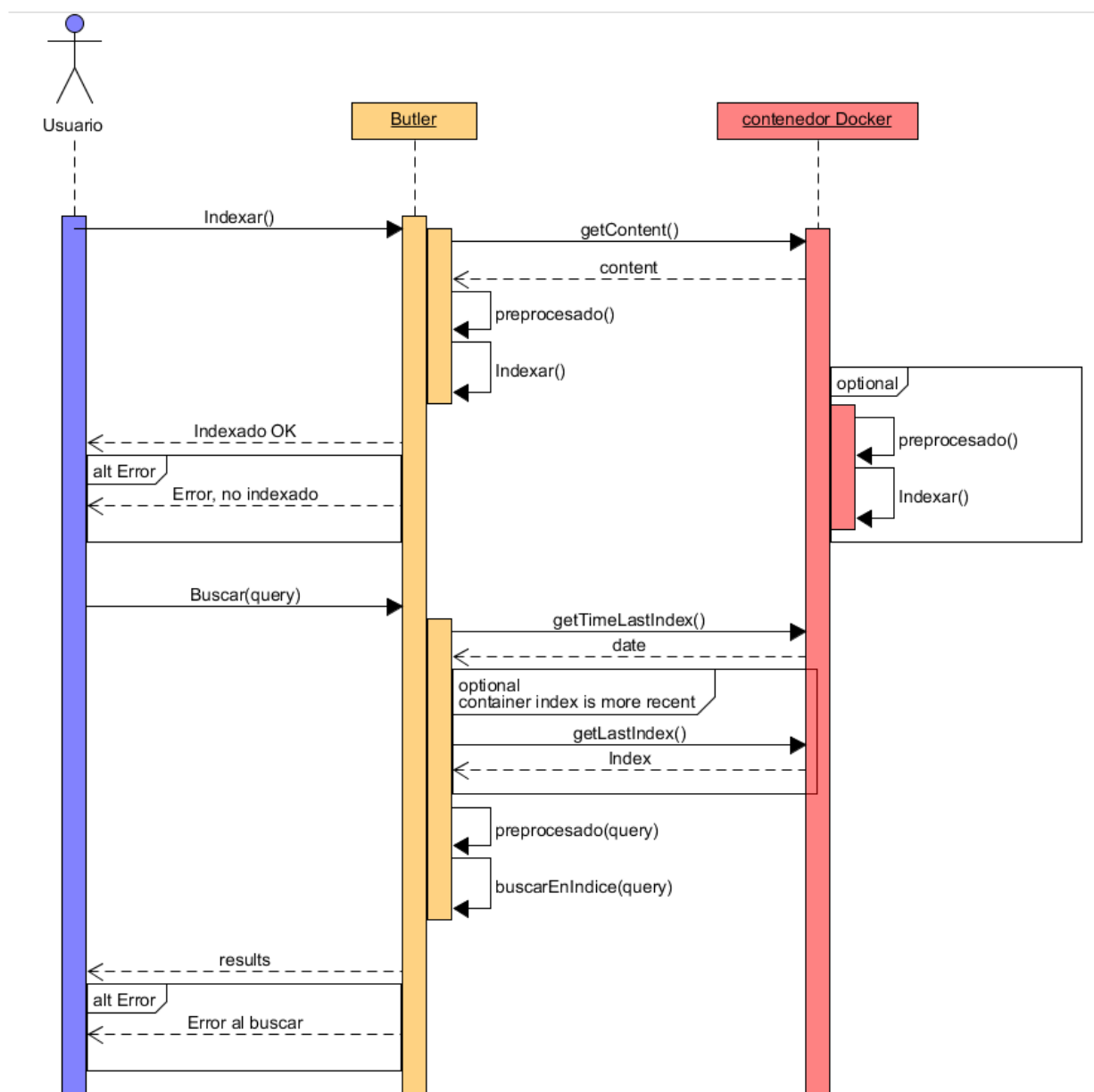


Figura 20: Diagrama de secuencia de una indexación

## Anexo E. Instalación Docker

Para la instalación y uso de Butler es necesaria la previa instalación de Docker.

La instalación es bastante sencilla y Docker tiene una muy buena documentación. Aquí se muestran los links necesarios.

Aquí se muestra una guía de instalación completa y configurando partes de Docker, si no, también están los links para una instalación rápida en linux, Mac, Windows.

Se recomienda la instalación completa, puesto que indica como configurarlo con mayor detalle y, por ejemplo, para dar permisos suficientes únicamente para Docker (Si no, puede haber algunos problemas con Butler).

Docker es una tecnología nueva, y durante la realización de este proyecto, se ha visto que Docker, a veces, si se tiene funcionando mucho tiempo, puede quedarse colgado (rara vez, pero ha ocurrido a veces) dando este mensaje la mayor parte de las veces (otras veces simplemente no había mensaje):

```
docker:      An      error      occurred      trying      to      connect:      Post
http://%2Fvar%2Frun%2Fdocker.sock/v1.22/containers/create:      read      unix      @-
>/var/run/docker.sock: read: connection reset by peer.
```

Para solucionar este problema (o algún otro relacionado con el mal funcionamiento de Docker), tan solo hay que ejecutar:

```
sudo service docker stop
sudo rm /var/lib/docker/network/files/local-kv.db
sudo service docker start
```

También puede dar este mensaje, el cual simplemente te avisa de que no está instalado correctamente o está mal configurado (aunque sea tan solo por falta de permisos suficientes):

'Cannot connect to the Docker daemon. Is 'docker daemon' running on this host?'

Existe la posibilidad de que aparezca este mensaje también, pero no es un problema, tan solo es un aviso, pero no impide el funcionamiento de Docker

```
WARNING: Error loading config file:/home/user/.docker/config.json - stat
/home/user/.docker/config.json: permission denied
```

Para solucionar esto se puede crear un fichero con un objeto Json vacío (('{}' como único contenido del fichero) para que el mensaje no vuelva a aparecer.