Facultad de Informática
UNIVERSIDADE DA CORUÑA

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN TECNOLOGÍAS DE LA INFORMACIÓN

Implantación completa de una web corporativa usando stacks

Estudiante: Rodrigo Villar Bello
Dirección: Carlos Fernández Lozano
Francisco Abel Cedrón Santaeufemia

A Coruña, 23 de junio de 2021.

A todos los que han estado ahí.

Agradecimientos

Primero de todo agradecer a mi padre y mi madrina por siempre estar detrás de mí, apoyándome y animándome a ser mejor cada día.

También agradecer a todos los muchachos y los chavales que me han acompañado durante esta etapa, tanto los nuevos como los viejos.

Por último, agradecer a Carlos por su labor durante este proyecto, ha sido un placer trabajar contigo.

Resumen

Los investigadores actuales necesitan automatizar más que nunca la publicación de resultados debido a que la gran cantidad de datos que manejan hace que este proceso sea lento y tedioso si se realiza de manera manual.

Dentro de este contexto, este proyecto se centra en la renovación del sitio web del grupo de investigación **RNASA-IMEDIR** junto con el despliegue de la infraestructura necesaria para sustentarlo utilizando tecnologías actuales que faciliten la publicación de resultados a los miembros del grupo, además del futuro mantenimiento.

Abstract

Today's researchers more than ever need to automate the publication of results because the sheer amount of data they handle makes this process slow and tedious if done manually.

Within this context, this project focuses on the renewal of the website of the **RNASA-IMEDIR** research group together with the deployment of the necessary infrastructure to support it using current technologies that facilitate the publication of results to the members of the group, as well as future maintenance.

Palabras clave:

- RNASA-IMEDIR
- Resultados dinámicos
- Stacks
- Shiny/R
- Jekyll
- Nginx
- Docker
- Docker-Compose
- Docker Swarm
- CentOS

Keywords:

- RNASA-IMEDIR
- Dynamic results
- Stacks
- Shiny/R
- Jekyll
- Nginx
- Docker
- Docker-Compose
- Docker Swarm
- CentOS

Índice general

Índice de figuras

Índice de tablas

Índice de códigos

Capítulo 1

Introducción

1.1 Motivación

Bien entrados en el siglo XXI, no hay campo o industria que se precie que no esté informatizada a día de hoy, y la investigación científica no es menos. En este tipo de investigaciones se realizan, con frecuencia, estudios o trabajos sobre diversos temas (*Big Data*, *Data Mining*, *Machine Learning* o *Blockchain*, por nombrar algunos de lo más conocidos actualmente) en los que intervienen una cantidad tan grande de datos que complica el trabajar con ellos, pudiendo incluso llegar a ser hasta inmanejables para un número pequeño de personas, dependiendo del alcance del trabajo. Y uno de los lugares donde este problema golpea con más fuerza es en la presentación de los resultados de estos estudios: unas pocas imágenes estáticas pueden no ser suficientes para transmitir una visión completa del análisis realizado.

Por estas razones, se hace necesario compartir los resultados de investigaciones de manera que se pueda interactuar con ellos de forma dinámica: de esta manera no sólo se consigue reducir enormemente la cantidad de tiempo y esfuerzo que los investigadores tienen que dedicar a la hora de presentar sus resultados, sino que, además, amplia y flexibiliza la capacidad de cualquier persona de validar, comprender y analizar los estudios realizados al exponer una mayor cantidad de datos y en un grano más fino que los que se podrían encontrar en una publicación de una revista de investigación.

Además, que estos resultados se publiquen junto con los datos de sus autores (ya sean grupos o individuos) ayuda a mantener y unificar la imagen corporativa de cualquier empresa y organismo como un laboratorio de investigación, enriqueciendo y proyectando hacia el exterior una imagen de unidad y seriedad.

Por último, es importante mencionar que, a pesar de que este tipo de acercamientos tiene ventajas (de las que ya se han hablado), estas no vienen sin un coste: la adición de nuevo software y servicios añade complejidad, elevando, en gran medida, el tiempo necesario en la gestión, administración y despliegue de la infraestructura. Por suerte, para solucionar este

problema existen tecnologías y herramientas actuales que facilitan el trabajo, implementando este tipo de sistemas utilizando *stacks*, es decir, colecciones de diferentes tipos de servicios que, al juntarlos, forman aplicaciones completas.

1.2 Objetivos

El objetivo principal de este proyecto es implementar una solución que unifique las publicaciones de los investigadores, tanto a nivel de grupo como a nivel individual. Para ello, se desplegarán diferentes plantillas web desarrolladas en **Jekyll** (más información en la sección 3.1) que recogerán información de cada investigador y del grupo en general. Además, se utilizarán diferentes Application Programming Interfaces (APIs) públicas, como Google Scholar o Publons, para recolectar ciertos datos (número de publicaciones, citas o revisiones, por ejemplo), pudiendo así, de una forma automatizada, mantener actualizada la información presentada en estos sitios web. Además, la web corporativa se encargará de enlazar con las webs de cada uno de los investigadores.

Por otro lado, la presentación de los resultados de los investigadores se realizará utilizando webs desarrolladas en **Shiny/R** (ver subsecciones 3.2.1 y 3.2.2). Debido a que la potencia a la hora de mostrar estos datos dinámicos en Shiny está directamente relacionada con el consumo de recursos (que, habitualmente, es elevado), se implementará un esquema basado en **Docker** (ver subsección 3.3.1) al cual se le añadirá una capa de **Docker Swarm** (ver subsección 3.3.2) para el balanceo de la carga de procesado. Aprovechando dicha infraestructura, los despliegues se realizarán con **Docker-Compose** (ver subsección 3.3.1) para automatizar la secuencia, generación y despliegue de elementos diferentes en el mismo proceso.

Por último, el acceso tanto a los diferentes sitios webs de investigadores como a las webs dinámicas de Shiny se realizará a través de **Nginx** (ver sección 3.4), que desempeñará los roles de servidor web y *proxy* inverso.

1.3 Estructura del documento

Esta memoria está estructurada en los siguientes apartados:

- Capítulo 1 — **Introducción**. Presentación del proyecto, de los objetivos que se pretenden cumplir y del propio documento

- Capítulo 2 — **Estado de la cuestión**. Se exponen algunas alternativas de los principales componentes que conforman el sistema y porqué no han sido seleccionadas para participar en este trabajo.

- Capítulo 3 — **Herramientas y tecnologías utilizadas**. Listado y descripción de los elementos principales que han intervenido en el proyecto.
- Capítulo 4 — **Metodología**. Explicación en detalle del método de trabajo utilizado, de su adaptación y del razonamiento detrás de su elección.
- Capítulo 5 — **Análisis de requisitos**. Listado y definición de los roles de los usuarios que usarán el sistema desarrollado y de sus requisitos.
- Capítulo 6 — **Planificación**. Se describen las distintas iteraciones durante las que se ha desarrollado el proyecto, de los recursos materiales necesarios y de los costes asociados.
- Capítulo 7 — **Desarrollo**. Descripción, en detalle, del trabajo realizado a lo largo del proyecto, dividido por iteraciones.
- Capítulo 8 — **Conclusiones**. Exposición del resultado final del proyecto en comparación con los requisitos iniciales, junto con las posibles mejoras a aplicar. Se cierra el capítulo listando qué asignaturas del grado han aportado valor de cara al desenvolvimiento de este trabajo.
- **Apéndices**. Dividido en:
  - Apéndice A — **Despliegue en CentOS**. Se tratan los distintos pasos necesarios para desplegar el proyecto en producción, además de la rotación de *logs* de los servicios y de algunas medidas que incrementan la seguridad del servidor utilizado.
  - **Lista de acrónimos**.
  - **Glosario**.
  - **Bibliografía**.

Capítulo 2

Estado de la cuestión

La búsqueda de soluciones *software* es siempre una tarea compleja: estas pueden estar pobremente diseñadas, pueden no ser lo suficientemente potentes para algunos de los casos de uso que se plantean, o pueden que no sean lo suficientemente flexibles de cara al futuro mantenimiento del sistema.

Entre los requisitos que se plantean de manera inicial en este proyecto, se pueden destacar los siguientes como los más importantes:

- Publicación de resultados de manera dinámica.
- Unificación de la imagen corporativa.
- Escalabilidad del sistema.

Un problema importante que se puede observar en estos requisitos es la dificultad de encontrar una única solución que cumpla con estos, es decir, que permita desplegar un sistema online que presente un *currículum* de un grupo de investigación y de sus miembros donde se puedan visualizar datos de sus estudios de manera dinámica y que, además, sea escalable.

Es por ello que se toma un enfoque más compartimentalizado a la hora de desarrollar este proyecto, dejando que distintas herramientas y tecnologías se ocupen de lo que mejor saben hacer, y que el sistema o aplicación resultante sea la combinación de las diferentes interacciones entre estas tecnologías. Este desacoplado permite poder actualizar o intercambiar estas aplicaciones individuales por otras sin que las demás se vean afectadas, además de posibilitar que el mantenimiento de la infraestructura sea dividido entre personas con distintos perfiles, dedicándose cada una a un solo tipo de aplicación o servicio.

A lo largo de este capítulo se comentarán algunas de las herramientas existentes que permiten resolver, en mayor o menor medida, los requisitos planteados, pero que no se han utilizado a lo largo de este proyecto, y se cerrará hablando del porqué de esta elección.

2.1 Generadores de sitios web estáticos

Los generadores de sitios web estáticos se encuentran dentro de una arquitectura denominada *Jamstack*, que intenta reducir la complejidad de un sitio web tradicional en el que suele correr un servidor web, una aplicación y una base de datos. Gracias al pregenerado de los ficheros servidos (principal característica de las herramientas que siguen esta arquitectura) se reduce el número de componentes necesarios para el funcionamiento de la aplicación, mejorando así la seguridad, la escalabilidad, el rendimiento, la portabilidad y la experiencia de desarrollo.

Como el número de tecnologías y herramientas generadoras de páginas web estáticas es demasiado grande como para hablar de todas ellas (la página web *jamstack.org* lista más de 325 de estas herramientas a junio de 2021), se hablará brevemente de algunas de las más importantes y conocidas.

2.1.1 Hugo

Hugo es una de las herramientas principales en este mercado. Es de código abierto, posee multitud de temas para tomar como base en el desarrollo de sitios web y su instalación se hace muy sencilla al agrupar todas sus dependencias en un mismo binario. Pero su principal punto fuerte es que está escrito en *Go*, lo que le proporciona una gran velocidad a la hora de construir páginas web respecto sus competidores, haciendo que sea conocido como "el *framework* de construcción de sitios web más rápido del mundo" [1] (ver figura comparativa 2.1).

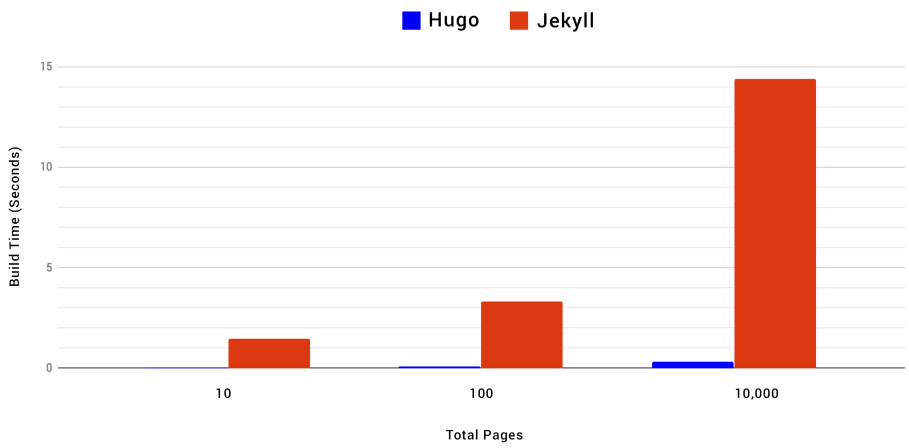


Figura 2.1: Comparación entre Hugo y Jekyll en la generación de sitios web.

Extraído de: https://forestry.io/blog/hugo-vs-jekyll-benchmark/.

2.1.2 Gatsby

Gatsby es un *framework* de código abierto basado en React, una librería JavaScript, y se centra en ofrecer lo mejor de los sitios web estáticos a la vez que elimina sus limitaciones, posibilitando la creación de sitios web dinámicos. Gatsby permite obtener datos de múltiples fuentes (como ficheros en formato *Markdown*, APIs, bases de datos, etc.) utilizando GraphQL, una lenguaje de *queries* que simplifica el proceso de construcción y optimización de las aplicaciones [2]. También posee temas y *plugins* como otras herramientas.

2.2 Servidores web y *proxies* inversos

Un servidor web es todo aquel software que recibe peticiones y responde a ellas sirviendo contenido de páginas web. Por otro lado, un *proxy* inverso es un servidor que permite acceder a múltiples servicios a través del mismo conjunto protocolo, dirección Internet Protocol (IP) y puerto, enrutando el tráfico de manera distinta internamente según las características de la petición recibida. Es común encontrarse con que estos últimos suelen tener funcionalidades adicionales como balanceo de carga, compresión, cacheo de contenido estático, cifrado Transport Layer Security (TLS), etc.

2.2.1 Apache HTTP Server

Apache HTTP Server es un servidor web de código abierto y multiplataforma que fue creado en creado en 1995 y es desarrollado por la Apache Software Foundation. En sus más de 25 años, Apache ha sido y sigue siendo uno de los servidores web más populares, lo que resulta en un gran soporte y en la existencia de muchas herramientas creadas alrededor de él [3].

Uno de los puntos fuertes que Apache ofrece por encima de algunas alternativas es la capacidad de utilizar un sistema de módulos de manera dinámica. Estos módulos, que extienden la configuración y capacidades básicas de Apache, pueden ser cargados o descargados (dependiendo de las necesidades) mientras que el servidor corre. Debido a la madurez del proyecto, es posible encontrar una gran cantidad de estos módulos que cubren y extienden necesidades como *logging*, cacheo, compresión, autenticación de clientes, *proxying*, etc. [4].

Un tipo de módulos que vale la pena destacar son los denominados Multi-Proccessing Modules (MPM), que permiten modificar la arquitectura del manejo de conexiones de Apache, flexibilizando las tareas de administración. Alguno de estos módulos son:

- **mpm_prefork** [5]. Implementa un servidor web que no utiliza hilos, es decir, cada proceso del servidor está dedicado enteramente a cada conexión Hypertext Transfer Protocol (HTTP), lo cual hace que sea muy rápido al no tener que lidiar con el manejo

de hilos. El término *prefork* que se encuentra en el nombre del módulo hace referencia a que Apache intenta mantener un conjunto de procesos ociosos y listos que puedan servir las peticiones entrantes con la mayor rapidez y no se tenga que esperar por el creado de estos subprocesos. Sin embargo, esto es un espada de doble filo: el tener que utilizar un proceso para cada petición hace que un gran número de conexiones concurrentes aumente el consumo de recursos en gran medida debido a que cada uno de estos procesos es una instancia completa del servidor Apache.

- **mpm_worker** [6]. El módulo *mpm_worker* implementa la gestión de hilos para servir peticiones entrantes, haciendo que, al contrario que el módulo *mpm_prefork*, cada proceso sirva múltiples conexiones HTTP, utilizando un hilo para cada una.

- **mpm_event** [7]. Este módulo es parecido al módulo *mpm_worker* en el sentido en que también utiliza hilos para responder al tráfico entrante, solo que el módulo *mpm_event* dedica un hilo para cada petición en lugar de para cada conexión HTTP. Esto supone que cada hilo se libere al terminar de servir la petición asignada a él, significando en una menor cantidad de hilos en total. Este manejo de las conexiones puede ser apropiado de cara a servir aplicaciones que tengan un alto *keepalive_timeout*, situación en la que el módulo *mpm_worker* no sería eficiente ya que mantendría los hilos desocupados, esperando por los datos que el cliente tenga que enviar.

2.2.2 Caddy

Caddy es otro servidor web de código abierto lanzado en 2015 que ha sido desarrollado en el lenguaje *Go*. Uno de sus puntos fuertes es la sencillez en su configuración: poniendo como ejemplo la configuración ofrecida por Jellyfin (un servidor multimedia de código abierto) en su documentación, podemos ver en la figura 2.2 como la configuración que hay que realizar para que Caddy actúe como *proxy* inverso es realmente sencilla y corta al compararla con la configuración que habría que realizar tanto en Nginx como en Apache.

Otro valor añadido de Caddy es la automatización en la configuración de Hypertext Transfer Protocol Secure (HTTPS): mientras que con otros servidores web habría que, por ejemplo, utilizar la herramienta *certbot* para configurar la utilización de conexiones seguras, con Caddy esto es prácticamente transparente para el administrador [8].

```
# Jellyfin hosted on http(s)://DOMAIN_NAME/jellyfin

server {
    listen 80;
    listen [::]:80;

    server_name DOMAIN_NAME;
    # You can specify multiple domain names if you want
    #server_name jellyfin.local;

    # use a variable to store the upstream proxy
    # in this example we are using a hostname which is resolved via DNS
    # (if you aren't using DNS remove the resolver line and change the variable to p
oint to an IP address e.g `set $jellyfin 127.0.0.1`)
    set $jellyfin jellyfin;
    resolver 127.0.0.1 valid=30;

    # Uncomment and create directory to also host static content
    #root /srv/http/media;
    index index.html;

    location / {
        try_files $uri $uri/ =404;
    }

    # Jellyfin
    location /jellyfin {
        return 302 $scheme://$host/jellyfin/;
    }

    location /jellyfin/ {
        # Proxy main Jellyfin traffic

        # The / at the end is significant.
        # https://www.acunetix.com/blog/articles/a-fresh-look-on-reverse-proxy-relat
ed-attacks/

        proxy_pass http://$jellyfin:8096/jellyfin/;

        proxy_pass_request_headers on;

        proxy_set_header Host $host;

        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-Host $http_host;

        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection $http_connection;

        # Disable buffering when the nginx proxy gets very resource heavy upon strea
ming
        proxy_buffering off;
    }
}
```

(a) Ejemplo de configuración de Jellyfin en Nginx.

```
<VirtualHost *:80>
    ServerName DOMAIN_NAME

    # Comment to prevent HTTP to HTTPS redirect
    Redirect permanent / https://DOMAIN_NAME

    ErrorLog /var/log/apache2/DOMAIN_NAME-error.log
    CustomLog /var/log/apache2/DOMAIN_NAME-access.log combined
</VirtualHost>

# If you are not using a SSL certificate, replace the "redirect"
# line above with all lines below starting with "Proxy"
<IfModule mod_ssl.c>
<VirtualHost *:443>
    ServerName DOMAIN_NAME
    # This folder exists just for certbot(You may have to create it, chown and chmod it to giv
e apache permission to read it)
    DocumentRoot /var/www/html/jellyfin/public_html

    ProxyPreserveHost On

    # Letsencrypt's certbot will place a file in this folder when updating/verifying certs
    # This line will tell apache to not to use the proxy for this folder.
    ProxyPass "/.well-known/" "!"

    ProxyPass "/socket" "ws://SERVER_IP_ADDRESS:8096/socket"
    ProxyPassReverse "/socket" "ws://SERVER_IP_ADDRESS:8096/socket"

    ProxyPass "/" "http://SERVER_IP_ADDRESS:8096/"
    ProxyPassReverse "/" "http://SERVER_IP_ADDRESS:8096/"

    SSLEngine on
    SSLCertificateFile /etc/letsencrypt/live/DOMAIN_NAME/fullchain.pem
    SSLCertificateKeyFile /etc/letsencrypt/live/DOMAIN_NAME/privkey.pem
    Protocols h2 http/1.1

    # Enable only strong encryption ciphers and prefer versions with Forward Secrecy
    SSLCipherSuite HIGH:RC4-SHA:AES128-SHA:!aNULL:!MD5
    SSLHonorCipherOrder on

    # Disable insecure SSL and TLS versions
    SSLProtocol all -SSLv2 -SSLv3 -TLSv1 -TLSv1.1

    ErrorLog /var/log/apache2/DOMAIN_NAME-error.log
    CustomLog /var/log/apache2/DOMAIN_NAME-access.log combined
</VirtualHost>
</IfModule>
```

(b) Ejemplo de configuración de Jellyfin en Apache HTTP Server.

```
example.com

reverse_proxy /jellyfin/* 127.0.0.1:8096
```

(c) Ejemplo de configuración de Jellyfin en Caddy.

Figura 2.2: Comparación de configuraciones de Jellyfin entre distintos servidores web.

Extraído de:

- https://jellyfin.org/docs/general/networking/nginx.html.
- https://jellyfin.org/docs/general/networking/apache.html.
- https://jellyfin.org/docs/general/networking/caddy.html.

2.3 Automatización de despliegues

Las herramientas de automatización de despliegues se encuentran dentro de lo que se conoce como Infrastructure as Code (IaC), y son tecnologías en las que su código fuente permite definir, aprovisionar y gestionar infraestructura de manera automatizada.

2.3.1 Podman y podman-compose

Dentro de la virtualización de contenedores, una buena alternativa es **podman-compose**, una implementación de docker-compose utilizando **Podman** como *backend*.

Uno de los principales dilemas que alguien se puede encontrar al utilizar Docker son los riesgos de seguridad que implica su demonio corriendo como usuario *root* [9].

Podman soluciona estos problemas de seguridad manteniendo, en la medida de lo posible, compatibilidad con Docker, siendo esto último posible al no ser más que otro *wrapper* alrededor de las mismas herramientas que Docker utiliza. Podman está diseñado para ser ejecutado *daemonless* y sin necesitar privilegios de *root*, además de aportar integración con systemd, cosa que Docker no ofrece [10].

A pesar de esto, Podman presenta algunos inconvenientes: herramientas externas que utilizan la API de Docker no funcionan con Podman, además de que podman-compose, a junio de 2021, todavía se encuentra en desarrollo.

2.3.2 Ansible

Saliendo de la virtualización, una de las principales herramientas que se pueden encontrar en la automatización de despliegues y configuración es **Ansible**, cuyo desarrollo comenzó en 2012 [11], y en 2015 fue adquirida por Red Hat [12]. Los puntos clave que han hecho que Ansible sea una herramienta exitosa a día de hoy se podrían resumir en los siguientes [13] [14]:

- Es **mínima**. Ansible sigue la filosofía de que las herramientas de gestión no deberían tener que imponer requisitos adicionales en el entorno en el que se ejecutan. Es por ello que es una tecnología *agentless* (esto es, no requiere de un demonio) que sólo necesita que un servidor Secure Shell (SSH) y Python estén instalados en los servidores a configurar.
- Es **segura**. La superficie de ataque es muy reducida debido a que Ansible está compuesta por un número muy pequeño de elementos.
- Es **fiable**. Si se escriben correctamente, las configuraciones realizadas por Ansible son idempotentes, esto es, realizar una acción múltiples veces supone el mismo resultado. De esta forma, si se conoce el código fuente utilizado por Ansible, también se conoce el estado en el que los servidores se encuentran.
- Es **fácil de usar**. Ansible utiliza un lenguaje declarativo basado en YAML Ain't Markup Language (YAML) y plantillas Jinja.

2.4 Balanceadores de carga

Los balanceadores de carga evitan la saturación de las aplicaciones dividiendo el tráfico de entrada entre diferentes réplicas del servicio.

Uno de los balanceadores de carga más populares es **HAProxy**: un servidor *proxy* y balanceador de carga de alta disponibilidad de código abierto escrito en C. Su primera versión fue lanzada en 2001 y, desde entonces, se ha convertido en una pieza fundamental en la arquitectura de muchas compañías de renombre como GitHub, Reddit o Instagram [15]. Esto último lo ha logrado gracias a su gran eficiencia, haciendo que pueda lograr hazañas como la de *forwardear* más de 2 millones de peticiones por segundo en una única instancia ARM [16].

Además de HAProxy, existen muchas otras tecnologías que proveen balanceamiento de carga, como por ejemplo, Kubernetes, Nginx, F5, Kemp, etc.

2.5 Conclusiones

Dentro de los perfiles de herramientas de los que se ha hablado a lo largo de este capítulo se pueden encontrar una gran cantidad de aplicaciones que cubren un buen rango de casos de usos que podrían ser potencialmente beneficiosos para el desarrollo de este proyecto. Desafortunadamente, ninguna de las que se ha comentado es la elección idónea en sus respectivos campos.

Por un lado, Hugo es una gran herramienta cuya principal ventaja frente a sus alternativas desgraciadamente no ofrece ningún valor en especial en este trabajo: no se van a crear sitios web con una gran cantidad de contenido para que se pueda beneficiar de su velocidad de construcción. De hecho, la ausencia de *plugins* en Hugo sí es algo que tener en cuenta (de manera negativa), ya que con una de sus alternativas, Jekyll (de la que se hablará en la sección 3.1), se pueden utilizar *plugins* como *jekyll-scholar*, que gestiona, de manera automática, todo lo relacionado con la generación de páginas que muestran la bibliografía de los investigadores.

Por otro, el uso de una tecnología como Gatsby sólo complicaría el desarrollo del proyecto de una manera innecesaria: a pesar de no tener la misma desventaja que Hugo al sí poseer *plugins*, Gatsby es excesivamente potente para lo que este trabajo necesita, por lo que su utilización significaría una etapa de aprendizaje y estudio previo que únicamente retrasaría el proyecto.

HAProxy sufre un problema similar a Gatsby: es demasiado compleja para realizar el balanceo de carga que se requiere, cosa que es extremadamente sencilla de realizar mediante Docker Swarm (subsección 3.3.2).

Dado que se utilizará Docker Swarm para balancear la carga, entonces se utilizará docker-compose (subsección 3.3.1) para la automatización del despliegue ya que las dos son herramientas que se encuentran dentro de lo que ofrece Docker, eliminando de la ecuación la posibilidad de usar Ansible. Además, a pesar de que podman-compose sí podría ser una alternativa real, que todavía se encuentre en desarrollo hace que su uso sea problemático al no poseer una paridad total de funcionalidades con docker-compose.

Por último, aunque tanto Apache como Caddy son servidores web que no presentan ningún elemento perjudicial en su utilización, se prefiere el uso de Nginx (más información en la sección 3.4) debido a que se posee experiencia previa, al contrario que con los otros dos servidores web.

Capítulo 3

Herramientas y tecnologías utilizadas

3.1 Generador de sitios web: Jekyll

Jekyll es una herramienta de línea de comandos que se ocupa de generar páginas web estáticas [17]. Es de código abierto, está escrito en Ruby y su principal atención se centra en varias ideas alrededor de su filosofía [18]:

- **Fácil de usar**. No se requiere una base de datos y el contenido del sitio web se crea a partir de ficheros en formato Markdown, Hypertext Markup Language (HTML) y Cascade Style Sheets (CSS), además de soportar la ingestión de datos a partir de ficheros YAML Ain't Markup Language (YAML), JavaScript Object Notation (JSON), Comma-sepparated values (CSV) y Tab-sepparated values (TSV). A pesar de ser simple, Jekyll ofrece cierto grado de potencia mediante el uso del motor de plantillas Liquid [19], similar a los usados en Django o Hugo.

- **Fácil de entender**. Jekyll se enorgullece de no ser una herramienta "mágica": hace lo que se le pide y nada más, lo que facilita que sus usuarios comprendan su funcionamiento interno.

- **Estable**. La evolución de la herramienta no debería romper sitios generados previamente y, en caso de que lo haga, se debería de facilitar a los usuarios el proceso de actualización.

- **Extensible**. Jekyll contiene sólo aquellas funcionalidades que utilizan la mayor parte de sus usuarios. Esto supone que también debe ser extensible, proporcionando muchas funcionalidades mediante *plugins*.

La extensibilidad de Jekyll brilla con lo que denomina *temas*, definido como "sistema que permite aprovechar estilos y plantillas mantenidas por la comunidad para personalizar la presentación de tu sitio" [20]. Esto quiere decir que existen páginas web ya hechas en cuanto a lo que la estructura y los estilos se refiere, esperando a que quién las quiera utilizar solo tenga que añadir su contenido (imágenes, textos, etc.) para que estén listas, aunque también se podrían modificar esas estructuras y estilos iniciales si se desea. Existen cientos de estos temas, tanto gratuitos como de pago, y se encuentran recogidos a lo largo de varias páginas web, como, por ejemplo, *jekyll-themes.com*.

Jekyll también facilita el migrado de blogs desde diferentes tipos de fuentes, como ficheros en múltiples formatos (Really Simple Sindication (RSS), CSV, etc.) o Content Management Systems (CMSs) (Drupal, Wordpress, etc.), entre otros. [21]

Por último, además de ser el motor detrás de GitHub Pages, Jekyll es usado por una gran cantidad de sitios web, siendo algunos de los más conocidos Bitcoin, Spotify for Developers o Ruby on Rails [22].

3.2 Aplicaciones web interactivas: R y Shiny

3.2.1 R

R es un lenguaje de programación y un entorno multiplataforma de software libre enfocado en la estadística computacional y gráficos [23]. R se encuentra entre los lenguajes estadísticos más potentes del mundo y es muy utilizado en el mundo de las ciencias debido a que provee de herramientas de análisis de datos interactivas y gráficos de alta calidad para diferentes campos de la investigación científica [24]. Además, R se sitúa entre los lenguajes de programación más populares a fecha de mayo de 2021, según múltiples fuentes (en concreto, TIOBE Index [25], PYPL [26] y RedMonk [27]).

R, licenciado bajo la versión 2 de la GNU's not Unix (GNU) General Public License (GPL), comenzó su desarrollo a principios de los 90 de la mano de Robert Ihaka y Robert Gentleman, integrantes, en aquel momento, del Departamento de Estadística de la Universidad de Auckland en Auckland, Nueva Zelanda. La creación de R supuso la existencia de una implementación alternativa de S, otro entorno y lenguaje de programación centrado en el análisis de datos y gráficos.

La naturaleza de R (recordemos, un lenguaje de programación) hace que sea sencillo extender su funcionalidad en *paquetes* para cualquiera que conozca el lenguaje, lo que permite la existencia de paquetes que implementan nuevas técnicas estadísticas especializadas, herramientas de creación de informes o de importación/exportación de datos, etc. Y entre la gran cantidad de funcionalidades extra existentes para R nos podemos encontrar Shiny.

3.2.2 Shiny

Shiny es un paquete de R de código abierto desarrollado por RStudio que permite crear aplicaciones web interactivas directamente desde R, sin necesidad de código HTML, CSS o JavaScript (JS). A continuación se enumeran algunas de las capacidades adicionales que ofrece Shiny (listado no exhaustiva, un listado completo se puede encontrar en su repositorio de GitHub):

- El modelo de programación reactiva de Shiny le permite tener que realizar una cantidad mínima de trabajo cuando un usuario interactúa con algún elemento de la aplicación que genera un cambio en la misma.
- Contiene widgets prediseñados, personalizables y fáciles de usar (gráficos, tablas, desplegables, etc.).
- Se integra con RMarkdown.

Además de la versión gratuita de Shiny, denominada Shiny Server Open Source, también existe una versión de pago, Shiny Server Pro, que añade funcionalidades como soporte de múltiples esquemas de autenticación, una consola para la gestión automatizada de las aplicaciones, securizado del servidor mediante el cifrado del tráfico, etc. [28] Desde el 1 de enero de 2021, Shiny Server Pro ha dejado de estar a la venta en pro del uso de RStudio Connect [29].

3.3 Virtualización ligera y balanceo de carga: Docker y Docker Swarm

3.3.1 Docker

Docker es una tecnología multiplataforma de código abierto de contenerización de aplicaciones que sigue una arquitectura de microservicios. Docker se ocupa de empaquetar software en unidades estandarizadas y autocontenidas, denominadas *contenedores Docker*, que ejecutan una configuración determinada definida con anterioridad en una *imagen* [30] [31].

Estos contenedores constituyen una alternativa a las máquinas virtuales a la hora de desplegar servicios debido a los beneficios que ofrecen, siendo los siguientes algunos de los más importantes [32] [33]:

- **Aplicaciones más ligeras**. Las imágenes contienen únicamente el código de la aplicación para la que fueron diseñadas, resultando en un menor tamaño que las máquinas virtuales.

- **Mayor eficiencia en la gestión de recursos**. Al contrario que las máquinas virtuales, los contenedores no ejecutan un sistema operativo completo, lo que provoca que la utilización de recursos (procesador, memoria, etc.) sea menor.
- **Portabilidad**. Las imágenes que ejecutan los contenedores son autocontenidas, es decir, contienen todo el código y las dependencias necesarias para correr la aplicación. Esto hace que los contenedores sean completamente independientes del *host* donde se ejecutan, garantizando así el mismo resultado en diferentes entornos.

En la figura 3.1 se muestra, de forma visual, una comparación entre máquinas virtuales y contenedores.

Popularidad

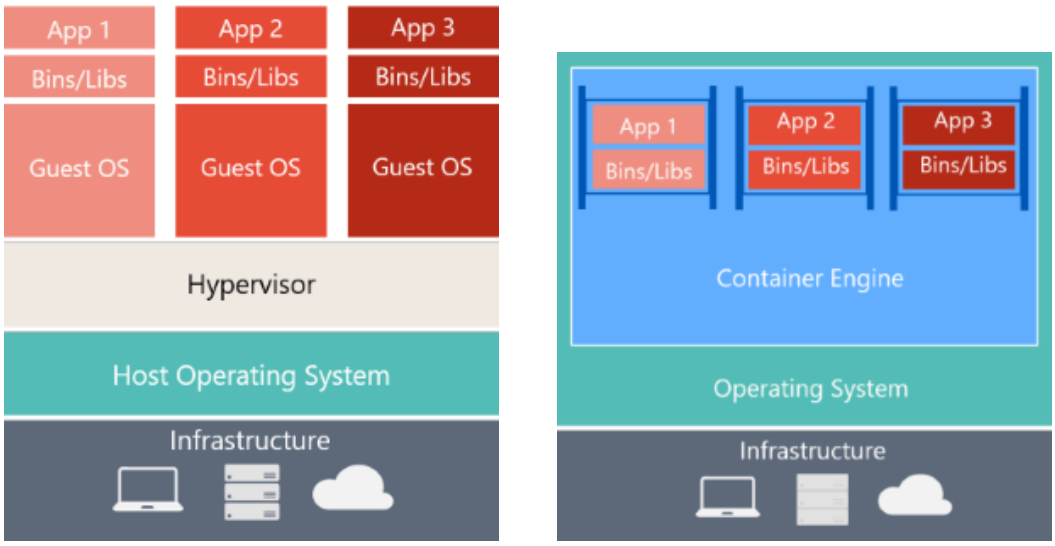
A pesar de ser una tecnología relativamente joven (su lanzamiento se produjo en 2013), Docker ya cuenta con una gran popularidad: más de 11 millones de desarrolladores y 7 millones de aplicaciones la utilizan a mayo de 2021 [34]. Esto también se puede apreciar en la encuesta llevada a cabo por StackOverflow en 2020, cuyos resultados muestran como Docker es una de las plataformas más populares entre desarrolladores, siendo únicamente superada por Linux y Windows [35].

Docker CE y Docker EE

En la actualidad, Docker existe en dos versiones: Docker Community Edition (CE) y Docker Enterprise Edition (EE). Mientras que la primera es totalmente gratuita, la segunda es la versión comercial, la cual está disponible en sistemas operativos certificados y proveedores en la nube, y está optimizada para despliegues críticos de negocios. Aun así, las dos versiones cuentan con las mismas características principales [36].

Docker-Compose

Docker permite el despliegue de aplicaciones multicontenedor mediante el uso de **Docker-Compose**, herramienta que ejecuta estos entornos a partir de su configuración definida, de forma declarativa, en ficheros YAML [37].



(a) Entorno con máquinas virtuales.

(b) Entorno con contenedores Docker.

Figura 3.1: Comparación de máquinas virtuales tradicionales con contenedores Docker.

Extraído de: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/container-docker-introduction/docker-defined.

Docker Hub

Aunque las imágenes utilizadas por los contenedores Docker pueden ser creadas desde 0, estas también se pueden generar a partir de otras imágenes. Docker cuenta con un repositorio de imágenes llamado **Docker Hub**, donde se pueden encontrar un gran número de estas imágenes (a mayo de 2021 se superan las 100.000 [38]), publicadas por vendedores software o por proyectos de código abierto o de la comunidad. Destacar que Docker Hub también permite el uso de repositorios privados para organizaciones y equipos, además de ofrecer generación automatizada de imágenes desde otras plataformas como GitHub o BitBucket [39].

3.3.2 Docker Swarm

Docker Swarm es una herramienta de orquestación de contenedores nativa de Docker, es decir, es una herramienta que se ocupa de gestionar, escalar y mantener aplicaciones contenerizadas [40]. Permite crear grupos de *hosts* que ejecutan Docker, denominados *swarms* o "enjambres", que, de forma conjunta, actúan como un *cluster* con el fin de proporcionar balanceo de carga para los servicios que corren. Además de este balanceo de carga, Docker Swarm provee alta disponibilidad y tolerancia a fallos: en caso de sufrir una caída, los contenedores del *host* caído se reinician en uno nuevo, intentando así mantener los servicios siempre en funcionamiento. Docker Swarm también puede escalar junto con el ciclo de vida de la aplicación: se puede comenzar con un *cluster* de un único *host* y escalar a un número mayor (o menor)

según las necesidades. De la misma forma que Docker-Compose, Docker Swarm utiliza ficheros en formato YAML para definir los componentes y configuraciones de las aplicaciones que componen el "enjambre" [37] [41].

3.4 Servidor web y *proxy* inverso: Nginx

Nginx es un servidor web de código abierto, aunque también puede usarse como *proxy* inverso, balanceador de carga o servidor *proxy* de email, entre otras funciones [42]. Nginx, que comenzó su desarrollo en 2002 a manos de Igor Sysoev, se creó con el objetivo de solucionar limitaciones en el rendimiento que los servidores web Apache sufrían en aquel momento [43]. A fecha de junio de 2021, Nginx es uno de los servidores web más utilizados [44].

Además de Nginx Open Source, existe también Nginx Plus, una versión comercial que añade capacidades de nivel empresarial como alta disponibilidad, controles de salud, descubrimiento Domain Name System (DNS), persistencia de sesiones y una API Representational state transfer (REST) [45].

3.5 Menciones honorables

A continuación se listan algunas herramientas y tecnologías que no han sido el principal foco de atención de este proyecto, pero que, aún así, merecen la pena ser mencionadas:

- **Python y Shell**. Estos lenguajes de *scripting* se han utilizado para implementar la obtención de datos de investigadores de APIs públicas.
- **HTML, CSS y Markdown**. Diferentes lenguajes de marcas han sido necesarios para modificar la estructura y estilizar los distintos sitios web.
- **Git**. El código creado se ha versionado utilizado Git como software de control de versiones.
- **Herramientas del desarrollador en distintos navegadores web**. Una buena parte del proceso de depuración de la configuración del servidor web y *proxy* inverso se ha llevo a cabo utilizando estas herramientas, siendo "red" la que más importancia ha cobrado.
- **Vagrant y VirtualBox**. El testeo del despliegue y las configuraciones del sistema creado se ha realizado utilizando máquinas virtuales gestionadas por estas dos herramientas.

Capítulo 4

Metodología

En el ámbito de la ingeniería software, las metodologías juegan un papel de gran importancia en el proceso de desarrollo software al definir unas pautas, reglas y prácticas que se deberían seguir si se quiere crear un buen producto software (que cumpla unos estándares de calidad o fiabilidad mínimos, por ejemplo) en una cantidad de tiempo y con unos costes razonables. Aunque no es estrictamente necesario utilizar una metodología para crear un buen producto, es recomendable analizar el proyecto a realizar con anterioridad para así poder estudiar si alguna de las metodologías existentes encaja en el proyecto y puede aportar un valor adicional en caso de ser utilizada.

Este proyecto presenta ciertas características que hacen que sea idónea la utilización de una metodología ágil como ***Scrum***, un marco de trabajo ligero que ayuda a las personas, los equipos y las organizaciones a generar valor mediante soluciones adaptativas para problemas complejos [46]. A continuación se mencionan algunas de estas características:

- El cliente participará a lo largo del desarrollo de todo el proyecto.
- Inicialmente, se requiere de una funcionalidad mínima usable sobre la que se podrán realizar modificaciones con posterioridad. Esto encaja con *Scrum*, una metodología incremental e iterativa.
- Existe la posibilidad de cambios en los requisitos en algún punto del desarrollo. Esto se puede solventar con relativa facilidad dado que, en *Scrum*, se trabaja en iteraciones de corta duración.

En las siguientes secciones se explica, con detalle, algunas de las características más importantes de la metodología *Scrum*, aunque un diagrama resumen se puede encontrar en la figura 4.1. También se comentará cómo se ha procedido a adaptar esta metodología en este trabajo.

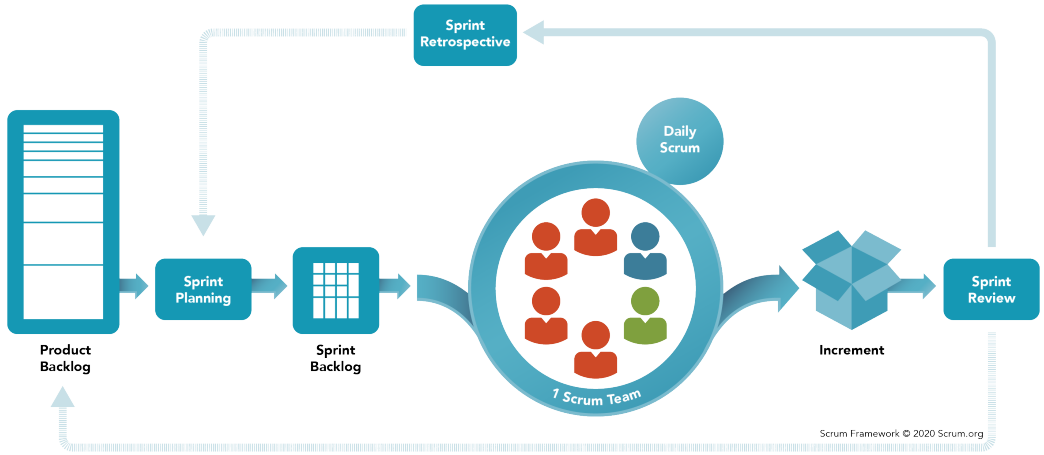


Figura 4.1: Diagrama del proceso *Scrum*.

Extraído de: https://www.scrum.org/resources/what-is-scrum.

4.1 *Scrum*

4.1.1 El Equipo *Scrum*

La unidad fundamental de *Scrum* es un pequeño equipo denominado Equipo *Scrum*. Este equipo es multidisciplinario (es decir, cuenta con todo lo necesario para llevar a cabo el proyecto que tenga asignado) y está autogestionado (decide qué trabajo realizar, quién lo hace, cómo y cuándo sin tener que depender de personas o grupos externos). Está compuesto de 3 importantes piezas: un **Propietario de Producto**, un **Equipo de Desarrollo** y un ***Scrum Master***.

Propietario de Producto

El Propietario de Producto es el responsable de maximizar el valor del producto resultante del trabajo del Equipo *Scrum*. Esto lo realiza creando, comunicando y ordenando por importancia aquello que es necesario para mejorar el producto, denominado Pila de Producto.

Es importante comentar que el Propietario de Producto es una única persona, no un comité, y tiene la última palabra en cuanto a qué tareas se deben realizar: el trabajo del que se ocupa el Equipo de Desarrollo está ligado a las decisiones del Propietario de Producto, por lo que querer modificar este trabajo supone convencer de ello a este último.

Equipo de Desarrollo

Está compuesto de aquellas personas que se ocupan de entregar un Incremento del producto "terminado" al final de cada *Sprint* (terminología de *Scrum* para referirse a una iteración), pudiendo, de forma ideal, desplegar estos Incrementos en producción.

El tamaño de este equipo no debe ser demasiado pequeño, para poder mantenerse ágil y no encontrarse con limitaciones en cuanto a las habilidades necesarias para llevar a cabo alguna tarea, ni tampoco demasiado grande, generando complejidad innecesaria durante el desarrollo. El tamaño óptimo suele encontrarse entre 3 y 9 personas, dependiendo de las idiosincrasias del proyecto.

Scrum Master

El *Scrum Master* es responsable de la efectividad del Equipo *Scrum* y debe promocionar y ayudar en el entendimiento de la teoría, práctica, reglas y valores de la metodología *Scrum*, además de eliminar cualquier impedimento en el progreso del Equipo *Scrum* y asegurarse de que los todos los eventos se llevan a cabo, son positivos y productivos, y no se extienden más de lo debido.

4.1.2 Eventos en *Scrum*

Los eventos definidos dentro del marco *Scrum* tienen el objetivo de mantener una regularidad, minimizar la necesidad de reuniones no definidas y crear oportunidades formales para la inspección y la adaptación en algún aspecto del desarrollo del producto según se crea conveniente. Los diferentes eventos en *Scrum* (que son períodos de tiempo finito, es decir, tienen una duración máxima), son los ***Sprints***, la reunión de **Planificación del *Sprint***, el ***Scrum* Diario**, la **Revisión del *Sprint*** y la **Retrospectiva del *Sprint***.

Sprints

Los *Sprints* son el núcleo de *Scrum* y de cada uno de ellos se obtiene un Incremento del producto. A continuación se presentan algunas de sus características que se ha considerado más importantes:

- Dentro de un *Sprint* están contenidos todos los demás evento, además del trabajo llevado a cabo por el Equipo de Desarrollo.
- No se debe modificar ni la duración ni los objetivos de un *Sprint* una vez comenzado.
- La finalización de un *Sprint* conlleva el comienzo de otro.

- Son eventos con una duración no mayor de un mes, y esta suele ser consistente a lo largo de todo el proceso de desarrollo.
- Se puede llevar a cabo la cancelación de un *Sprint* en caso de que su objetivo pase a estar obsoleto, pero sólo el Propietario del Producto tiene la potestad para realizar esta cancelación. A pesar de esto, la cancelación de un *Sprint* no se suele llevar a cabo debido a su corta duración.

Planificación del *Sprint*

La Planificación del *Sprint* es el primer evento que ocurre en un *Sprint*, tiene una duración máxima de 8 horas y en él el Equipo *Scrum* planifica el trabajo a realizar durante esa iteración. En esta reunión de planificación se resuelven las preguntas "¿Qué se puede hacer en este *Sprint*?" y "¿Cómo se conseguirá completar el trabajo seleccionado?", obteniendo, de esta manera, el Objetivo del *Sprint*, una meta a cumplir durante la duración del *Sprint* que ayuda a que el Equipo de Desarrollo entienda el porqué del Incremento que se construye.

Scrum Diario

El *Scrum* Diario es un evento con una duración aproximada de 15 minutos que es llevado a cabo de forma diaria por el Equipo de Desarrollo y tiene como objetivo centrarse en el Objetivo del *Sprint* para obtener el plan a cumplir durante el próximo día, teniendo que adaptar este plan según el progreso que se haya logrado hasta el momento.

Destacar que el *Scrum* Diario no tiene porqué ser el único momento donde el Equipo de Desarrollo se reúne para evaluar su situación y ajustar el plan a seguir, sino que esto también puede ocurrir a lo largo de la jornada si se ve necesario.

Revisión del *Sprint*

Al final del *Sprint* se produce este evento, una reunión de un máximo de 4 horas donde el Equipo *Scrum* inspecciona el Incremento creado y determina los detalles de los elementos que se cree que pueden optimizar el valor del producto en desarrollo, adaptando, de esta manera, la Pila de Producto.

Retrospectiva del *Sprint*

La Retrospectiva del *Sprint* es un evento realizado después de la Revisión del *Sprint* y en él el Equipo *Scrum* se autoevalúa con el fin de mejorar de cara al siguiente *Sprint*. Se tratan temas como los individuos, los procesos o las herramientas para así identificar y priorizar aquellos cambios que se piense que puedan aportar un mayor valor de cara al futuro desarrollo del producto. Por último, la duración de la Retrospectiva del *Sprint* no debería superar las 3 horas.

4.1.3 Artefactos en *Scrum*

Los Artefactos *Scrum* son elementos que representan el valor y el trabajo, y están diseñados para maximizar la transparencia de la información clave. Los Artefactos *Scrum* son la **Pila de Producto**, la **Pila del *Sprint*** y el **Incremento**.

Pila de Producto

La Pila de Producto es una lista compuesta de todo aquello que se considera importante de cara a la mejora del producto, ya sean funcionalidades, requisitos, mejoras, correcciones o características acerca de los cambios a realizar en futuras iteraciones del desarrollo del producto. Se considera que esta lista es dinámica e incompleta dado que a medida que el producto es utilizado y desarrollado, este proporciona retroalimentación, haciendo que se produzcan modificaciones tanto en las descripciones de los elementos de la Pila de Producto como en su número, en la prioridad asignada, etc.

Pila de *Sprint*

Este artefacto está compuesto de:

- El **Objetivo del *Sprint***.
- Un **subconjunto** de elementos de la **Pila de Producto**.
- Un **plan** de entrega del **Incremento**.

Es decir, estos elementos le proporcionan al Equipo de Desarrollo las respuestas a las preguntas "¿Qué hay que hacer?" (gracias al subconjunto de la Pila de Producto), "¿Cómo hay que hacerlo?" (gracias al plan de entrega del Incremento) y "¿Por qué hay que hacerlo?" (gracias al Objetivo del *Sprint*). Esto ayuda a que se tenga una imagen completa del trabajo que ocurrirá a lo largo del *Sprint*.

Incremento

El Incremento es el resultado de añadir al anterior Incremento existente aquellos elementos de la Pila de Producto que han sido completados durante el *Sprint*. Estos Incrementos deben ser usables independientemente de que el Propietario del Producto quiera proceder a su despliegue en producción o no.

4.2 Adaptación de la metodología

A pesar de que la metodología ágil *Scrum* tiene características que hacen adecuado su uso en este proyecto, se ha visto necesario realizar algunas modificaciones en las pautas que plantea.

La primera diferencia respecto *Scrum* es que dos personas, los directores del trabajo, han cumplido con el rol de Propietario de Producto, cuando lo recomendable es que este sea desempeñado por una sola. Por otro lado, el Equipo de Desarrollo ha estado compuesto exclusivamente por el alumno, lo cual hace que también difiera del tamaño recomendado de la guía oficial de *Scrum* (entre 3 y 9 personas) [46]. Para terminar con los roles, no se ha ejercido el de *Scrum Master* en este proyecto: este es un rol que debería cumplir el alumno, pero debido a su complejidad, llevarse a cabo supondría tener unas capacidades y habilidades que el alumno no posee, por lo que se ha optado por no desempeñarlo.

Por otro lado, se han desarrollado 3 *Sprints* de, aproximadamente, 1 mes de duración. El Equipo *Scrum* se ha reunido 1 vez al mes, y han sido en estas reuniones donde se han realizado tanto las Revisiones de los *Sprint* como su Planificación. Terminando con los artefactos en *Scrum*, en las Planificaciones de *Sprint* se ha obtenido la Pila de Producto a partir de los requisitos a cumplir en ese *Sprint*, resultando en una lista de Historias de Usuario (en la sección 5.2 se puede encontrar más información).

Capítulo 5

Análisis de requisitos

5.1 Roles

A continuación se definen los diferentes tipos de perfiles de los usuarios para los que se crea el producto:

- **Investigador**. Este rol constituye los principales usuarios del sistema a desarrollar, y se ocupará de actualizar su propia página web y crear aplicaciones Shiny que le permitan publicar resultados de sus estudios de manera dinámica.

- **Administrador del sitio web corporativo**. En este rol se encuentran aquellas personas que se ocuparán de actualizar los datos que se mostrarán en la página web corporativa (datos acerca de las líneas de investigación, de los miembros del grupo, nuevas noticias, etc.).

- **Administrador de sistemas**. El sistema a desarrollar no es una aplicación corriente que se instale en un portátil u ordenador de escritorio, sino que es necesario que se encuentre corriendo en un servidor expuesto a Internet. Esto supone la existencia de un nuevo rol junto con un nuevo tipo de requerimientos relacionados con el despliegue, configuración y administración del sistema creado y del servidor donde este sistema se despliega.

5.2 Historias de usuario

5.2.1 Pila de Producto

La Pila de Producto no es más que una lista de Historias de Usuario, esto es, las funcionalidades deseadas vistas desde la perspectiva del usuario que las va a utilizar. Estas historias se

HU	Descripción
H01	Como I quiero una página web orientada a mi perfil de investigación.
H02	Como AS quiero que los servicios a desplegar estén virtualizados en Docker.
H03	Como AS quiero que se acceda a Shiny utilizando Nginx como *proxy* inverso.
H04	Como I quiero publicar resultados de investigaciones de manera dinámica.
H05	Como I quiero actualizar información del investigador recogida de APIs públicas.
H06	Como AS quiero flexibilidad a la hora de configurar de Nginx.
H07	Como AS quiero rotación de *logs* para los servicios a desplegar.
H08	Como AS quiero aumentar la seguridad del servidor donde se desplegarán los servicios.
H09	Como AS quiero poder utilizar Docker Swarm como capa de balanceo en la virtualización con Docker.
H10	Como AW quiero una página web orientada a un perfil corporativo.
H11	Como I/AW quiero que la imagen corporativa del grupo de investigación esté unificada.
H12	Como AS quiero configurar y desplegar, de manera sencilla, los sitios web en producción.
H13	Como AW quiero mostrar en la página web corporativa la bibliografía de los investigadores de manera conjunta.

Tabla 5.1: Historias de usuario en la Pila de Producto.

suelen escribir de la siguiente manera: "Como *<tipo de usuario>* quiero *<aquello que se busca>* para que así *<resultado que se quiere lograr>*".

En la figura 5.1 se muestran las historias de usuario que se han tratado durante el desarrollo del proyecto. Por motivos de tamaño, el rol de *Investigador* será representado como "I", el rol de *Administrador del sitio web corporativo* como "AW" y el rol de *Administrador de sistemas* como "AS".

5.2.2 Funcionalidades

En este apartado se recoge una breve descripción de cada una de las Historias de Usuario vistas en la subsección anterior.

H01 — Página web orientada al perfil de investigación

Se buscará una tema de Jekyll que esté orientado a perfiles de investigadores, facilitando la publicación de distintos tipos de bibliografía.

H02 — Desplegar servicios en Docker

Se creará la configuración necesaria para desplegar los diferentes servicios (tanto configuración propia de los servicios como configuración de Docker) y se comprobará que la interacción con ellos se produzca de la manera que se espera una vez puestos en ejecución.

H03 — Utilizar Nginx como *proxy* inverso para acceder a Shiny

Se restringirá el acceso a Shiny desde el *host* y sólo se permitirá a través de Nginx. Para realizar esto se requiere configurar Nginx como *proxy* inverso.

H04 — Publicar resultados de investigación de manera dinámica

Desplegar aquella infraestructura necesaria para permitir a los investigadores la publicación de resultados de manera dinámica, utilizando Shiny.

H05 — Obtener datos de APIs públicas

Se codificará un *script* o programa que automatice la recogida de datos de los investigadores de Google Scholar y Publons.

H06 — Flexibilizar la configuración de Nginx

Aprovechando la imagen de Docker con la que se desplegará el servicio Nginx, se flexibilizará su configuración para dar al Administrador de sistemas una mayor cantidad de posibilidades y una mayor rapidez a la hora de modificar la configuración de Nginx, obviando la necesidad de cambiar el propio código fuente.

H07 — Rotación de *logs* de los servicios desplegados

Se configurará la rotación de los *logs* generados por los diferentes servicios creados, evitando así la saturación del sistema.

H08 — Securización del servidor

Se evaluarán e implementarán diferentes medidas que aumenten la seguridad del servidor expuesto a Internet.

H09 — Balancear la carga de los servicios con Docker Swarm

Se modificará la configuración existente del despliegue en Docker para permitir el balanceo de carga de los servicios.

H10 — Página web orientada al perfil corporativo

Se buscará un tema de Jekyll que sea adecuado como *currículum* online de un grupo de investigación.

H11 — Unificar la imagen corporativa del grupo de investigación

Se permitirá el rápido acceso a los sitios webs de los diferentes miembros del grupo desde la propia página del grupo de investigación.

H12 — Configurar y desplegar fácilmente los sitios web en producción

Se realizará la configuración necesaria en Docker para que se configure, de manera automática, con los parámetros requeridos para el despliegue en producción.

H13 — Mostrar la bibliografía de los investigadores en el sitio web corporativo

Se permitirá la visualización desde la página web corporativa de la bibliografía conjunta de los investigadores del grupo.

Capítulo 6

Planificación

En este apartado se detalla tanto la planificación del proyecto, como los recursos y los costes involucrados en el desarrollo.

Destacar también que no se ha visto necesario detallar el seguimiento del proyecto debido a la ausencia de desvíos durante el desarrollo del mismo. Esto es resultado del conocimiento previo que se poseía con anterioridad al comienzo de este trabajo, facilitando enormemente todas aquellas tareas relacionadas con la administración de sistemas en general.

6.1 Iteraciones

6.1.1 *Sprint* 1

La iteración de arranque del proyecto comprende la búsqueda del tema de Jekyll, que dará vida a las páginas web de los diferentes investigadores y, una vez encontrado, este se configurará de acorde a los datos de un investigador perteneciente al grupo. También se configurará tanto los servicios Shiny y Nginx como su despliegue en Docker, permitiendo así que este *Sprint* termine con una versión mínima, pero funcional, de la publicación de los resultados de estudios e investigaciones.

En la figura 6.1 se puede ver el diagrama de Gantt que representa este *Sprint*.

6.1.2 *Sprint* 2

Esta iteración abarca varias tareas relacionadas con la configuración desde el punto de vista de un administrador de sistemas. Estas tareas están orientadas a facilitar el trabajo del administrador, ya sea flexibilizando la configuración de Nginx para que la modificación de sus parámetros sea lo más sencilla posible, o securizando el servidor ante posibles amenazas externas o internas. Antes de comenzar con estas tareas administrativas, se termina la parte de la configuración de los sitios web individuales obteniendo aquellos datos relevantes de las

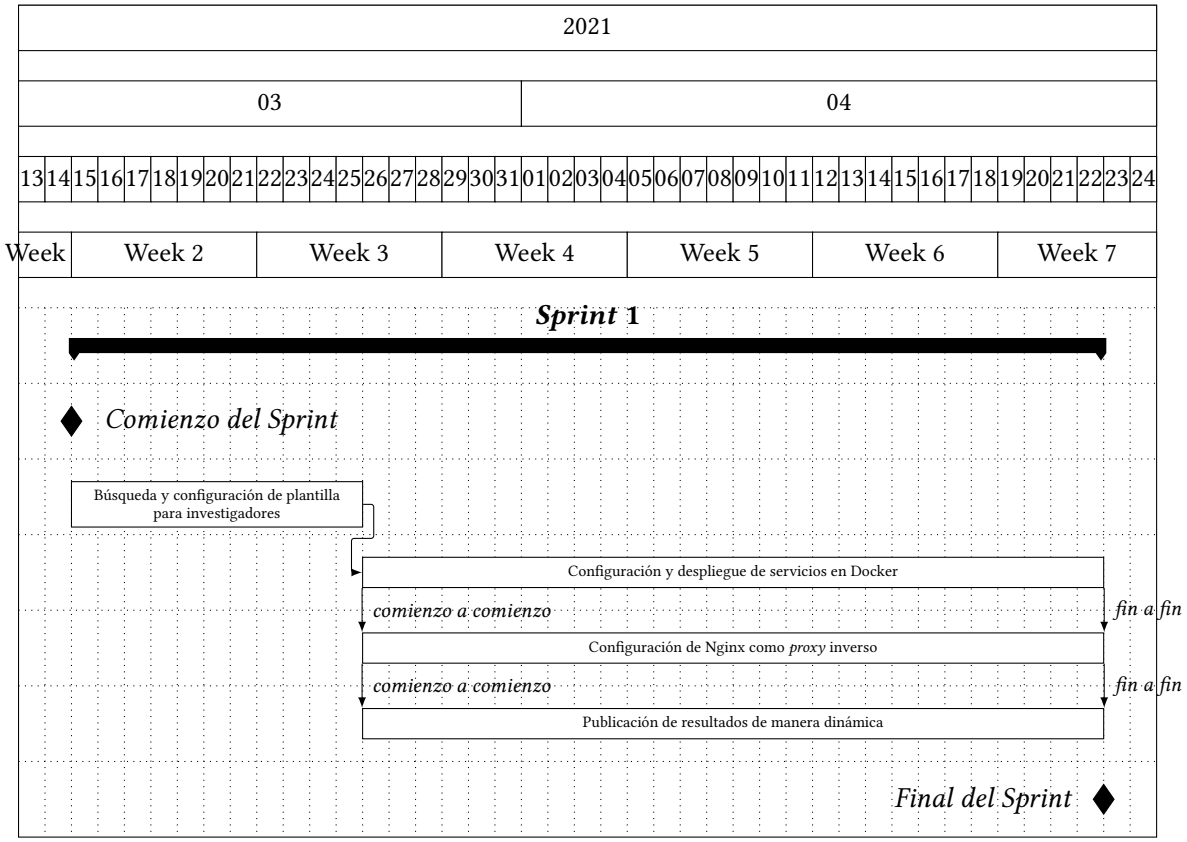


Figura 6.1: Diagrama de Gantt de la Planificación — *Sprint* 1.

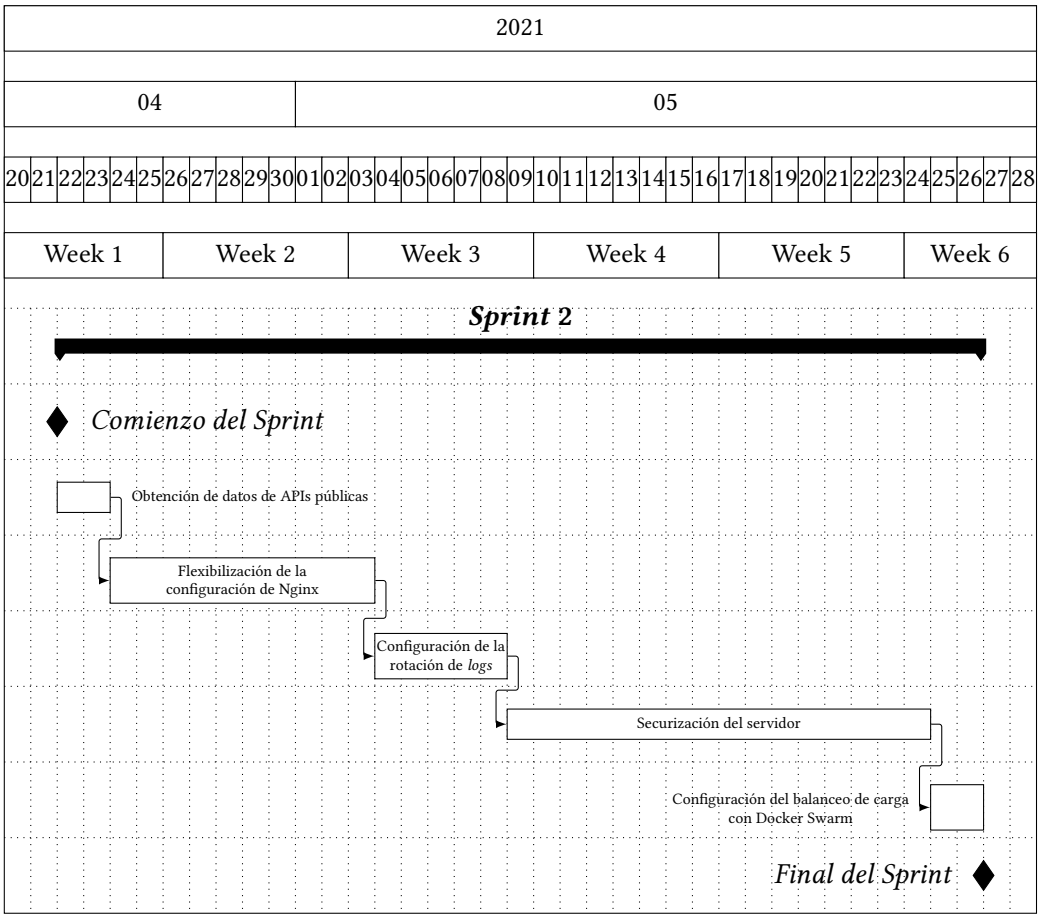


Figura 6.2: Diagrama de Gantt de la Planificación — *Sprint* 2.

distintas APIs.

De nuevo, este apartado está acompañado de un diagrama de Gantt que representa las tareas de las que se compone (figura 6.2).

6.1.3 *Sprint* 3

La última iteración del proyecto termina con el desarrollo de la página web corporativa: búsqueda de un tema en Jekyll adecuado, configuración y unificación de la imagen corporativa. También se realizan aquellos cambios necesarios para adaptar los servicios a un entorno en producción. El diagrama de Gantt de esta iteración se puede encontrar en la figura 6.3.

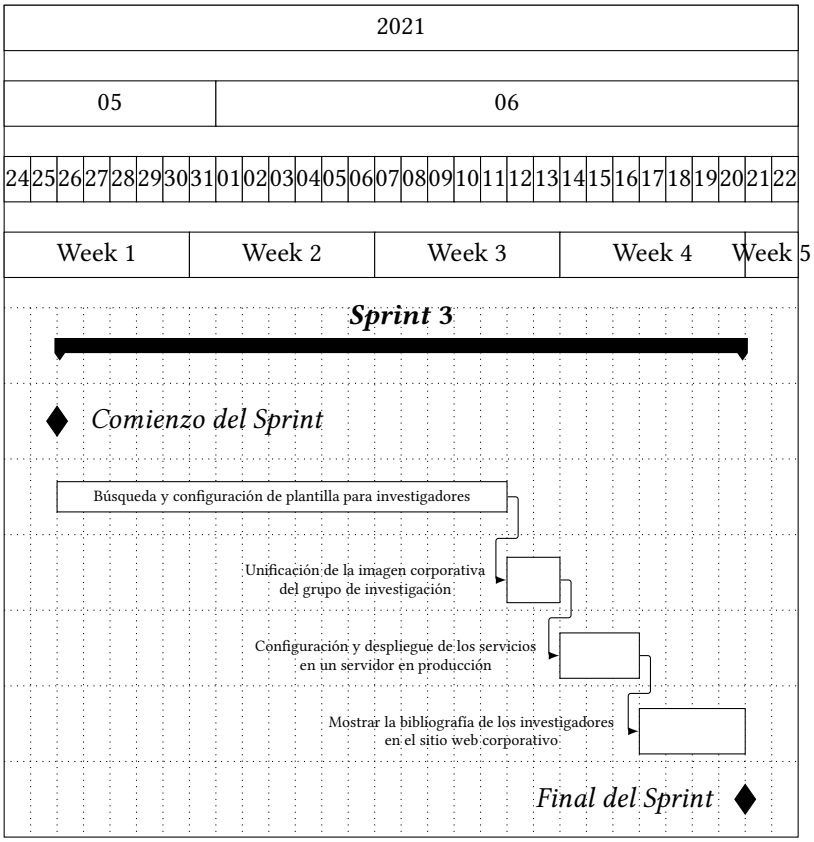


Figura 6.3: Diagrama de Gantt de la Planificación — *Sprint* 3.

6.2 Recursos y costes

6.2.1 Recursos y costes humanos

Para comenzar, los 3 recursos humanos que han intervenido en el proyecto han sido:

- 1 alumno que, en su mayor parte, ha desempeñado un rol de programador durante unas 280 horas. El sueldo de un programador junior en España en junio de 2021 se estima alrededor de los 19.000 € anuales (según la web Indeed [47]), lo que supone un coste por hora de *sueldo anual* / (*meses en un año* * *horas trabajas por mes*) = 19.000 € / (12 meses * 148 horas/mes) ≈ 10,70 €.

- 2 directores, desempeñado un papel de jefes de proyecto durante 30 horas cada uno. El sueldo de un jefe de proyecto en España en junio de 2021 se estima alrededor de los 39.000 € anuales (de nuevo, según la web Indeed [48]), lo que supone un coste por hora de *sueldo anual* / (*meses en un año* * *horas trabajas por mes*) = 39.000 € / (12 meses * 148 horas/mes) ≈ 21,96 €.

En la tabla 6.1 se puede observar el desglose de estos costes.

	Coste/hora	**Horas trabajadas**	**Coste total**
Programador	10,70 €	280	2.996,00 €
Jefe de proyecto 1	21,96 €	30	658,80 €
Jefe de proyecto 2	21,96 €	30	658,80 €
Total	**54,62 €**	**340**	**4.313,60 €**

Tabla 6.1: Costes humanos del proyecto.

6.2.2 Recursos y costes materiales

En cuanto a los recursos materiales, estos se pueden dividir en *hardware* y *software*.
Los dos recursos materiales *hardware* utilizados en el proyecto han sido:

- Un portátil personal donde se ha desarrollado el proyecto. Dado que tiene un valor estimado de 600,00 €, su vida útil es de, aproximadamente, 4 años y el proyecto se ha realizado a lo largo de 13 semanas, el coste del equipo ha sido de *duración del proyecto en meses* * *coste del equipo por mes* = 3,25 meses * (600,00 € / 48 meses) = 40,63 €.

- El servidor donde se ha realizado el despliegue del sistema creado. Este servidor tiene unas características parecidas al *PowerEdge R6515 Rack Server* de Dell, cuyo coste es de 1.809,27 € [49].

	Coste
Portátil	40,63 €
Servidor	1.809,27 €
Total	**1.849,90 €**

Tabla 6.2: Costes materiales *hardware* del proyecto.

En la tabla 6.2 se pueden observar el desglose de estos costes.

En cuanto a los costes materiales *software*, como se puede ver en la tabla 6.3, todo el *software* utilizado ha sido gratuito.

	Coste
Jekyll	00,00 €
Nginx (servidor web)	00,00 €
Nginx (*proxy* inverso)	00,00 €
Shiny	00,00 €
Docker	00,00 €
CentOS	00,00 €
Otro *software*	00,00 €
Total	**00,00 €**

Tabla 6.3: Costes materiales *software* del proyecto.

Sumamos el coste material *hardware* total con el coste material *software* total para obtener el coste material total (tabla 6.4).

	Coste
Costes materiales *hardware*	1.849,90 €
Costes materiales *software*	00,00 €
Total	**1.849,90 €**

Tabla 6.4: Costes materiales del proyecto.

6.2.3 Costes totales

Por último, para calcular el coste total del proyecto no hay más que sumar los costes humanos más los costes materiales. En la tabla 6.5 se puede apreciar que el coste final ha sido de **6.163,50 €**.

	Coste total
Costes humanos	4.313,60 €
Costes materiales	1.849,90 €
Total	**6.163,50 €**

Tabla 6.5: Costes totales del proyecto.

Capítulo 7

Desarrollo

En este capítulo se hablará de como se han desarrollado las diferentes historias de usuarios (definidas en el capítulo 5) a lo largo de los distintos *Sprints* del proyecto. El resultado del desarrollo se puede apreciar accediendo al sitio web desplegado en https://vi214.udc.es.

7.1 *Sprint* 1

7.1.1 H01 — Página web orientada al perfil de investigación

La plantilla de Jekyll que se ha usado como base para las páginas web de los investigadores es al-folio, y su principal característica es que está orientada a académicos: provee de un *layout* que, apoyándose en el *plugin jekyll-scholar*, genera una página adaptada para el listado de bibliografía, incluyendo información como ficheros Portable Document Format (PDF), *abstracts*, Digital Object Identifiers (DOIs), y más.

En las figuras 7.1, 7.2 y 7.3 se muestran algunos ejemplos de las principales partes del sitio web del investigador.

Figura 7.1: Página inicial del sitio web del investigador.

Figura 7.2: Muestra de las publicaciones del sitio web del investigador.

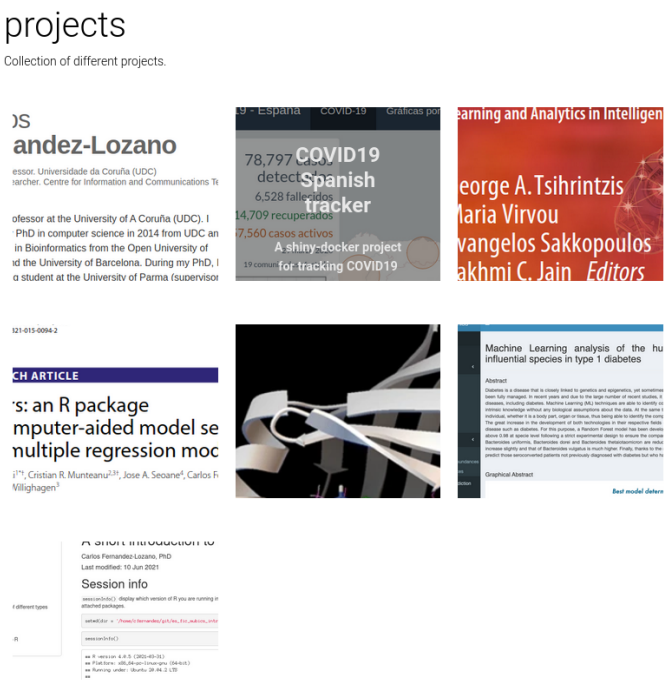


Figura 7.3: Proyectos del sitio web del investigador.

7.1.2 H02 — Desplegar servicios en Docker

Uno de los principales problemas de Docker es que sus contenedores, de manera predeterminada, utilizan el usuario *root*. Esto es posible solucionarlo, pero para ello hemos de modificar los permisos de algunos ficheros y directorios a los que los servicios a desplegar acceden (ver códigos 7.1 y 7.2).

```
FROM nginx:1.19.10

RUN rm --recursive --force /usr/share/nginx/html/*

COPY --chown=nginx:root  ./config/nginx/ /etc/nginx/
COPY --chown=root:root   ./sites/        /usr/share/nginx/html/

RUN mkdir --parent /data/nginx/cache    && \
    chown --recursive nginx:root           \
        /etc/nginx/                        \
        /var/cache/nginx/                  \
        /data/nginx/cache/              && \
    chmod --recursive u=rwX,g=rwX,o=rX     \
        /etc/nginx/                        \
        /var/cache/nginx/                  \
        /data/nginx/cache/              && \
    chmod --recursive u=rwX,g=rX,o=rX      \
        /usr/share/nginx/html/

USER nginx
```

Código 7.1: Dockerfile personalizado de Nginx.

```
FROM rocker/shiny:4.0.5

RUN rm --recursive --force /srv/shiny-server/*

COPY --chown=root:root ./config/shiny-server/  /etc/shiny-server/
COPY --chown=root:root ./shiny-apps/           /srv/shiny-server/

RUN chown --recursive shiny:shiny          \
        /home/shiny/                       \
        /var/log/shiny-server              \
        /var/lib/shiny-server          && \
    chmod --recursive u=rwX,g=rX,o=rX      \
        /etc/shiny-server/                 \
        /srv/shiny-server/

USER shiny
CMD ["/usr/bin/shiny-server"]
```

Código 7.2: Dockerfile personalizado de Shiny.

Una vez se tienen las imágenes deseadas, no hay más que utilizar docker-compose para desplegar los servicios (ver código 7.3).

```
version: '3'
services:
  web:
    image: nginx-custom:1.19.10
    ports:
      - 80:8080
      - 443:8443

  shiny:
    image: shiny-custom:4.0.5
```

Código 7.3: Contenido del fichero docker-compose.yml.

7.1.3 H03 — Utilizar Nginx como *proxy* inverso para acceder a Shiny

Configurar Nginx como *proxy* inverso es tarea sencilla: simplemente se ha de especificar a qué Uniform Resource Identifier (URI) se han de dirigir las peticiones para reenviarlas hacia el *backend* (en este caso, Shiny).

Los pasos realizados para implementar esta configuración han sido:

1. Si una petición es realizada al URI "/shiny-apps", este se modifica, pasando a ser "/shiny-apps/".

2. Si el URI de una petición comienza por "/shiny-apps/", este se reenvía a "shiny:3838", *host* y puerto donde el servidor Shiny está escuchando. Destacar que antes de realizar este reenvío, el URI se modifica (eliminando esa primera parte de "shiny-apps"), evitando así que las peticiones que llegan al servidor Shiny se hagan al *path* incorrecto.

En la figura 7.4 puede verse la implementación de estos cambios en la configuración de Nginx.

```
rewrite ^/shiny-apps$ $scheme://$http_host/shiny-apps/ permanent;

location /shiny-apps/ {

    rewrite ^/shiny-apps/(.*)$ /$1 break;
    proxy_pass http://shiny:3838;
}
```

Figura 7.4: Configuración de Nginx como *proxy* inverso.

7.1.4 H04 — Publicar resultados de investigación de manera dinámica

Una vez configurado Nginx como *proxy* inverso y desplegados los servicios en Docker se podrá acceder a las aplicaciones Shiny utilizando el URI que se ha especificado en la subsección anterior, como se puede ver resaltado en la figura 7.5.

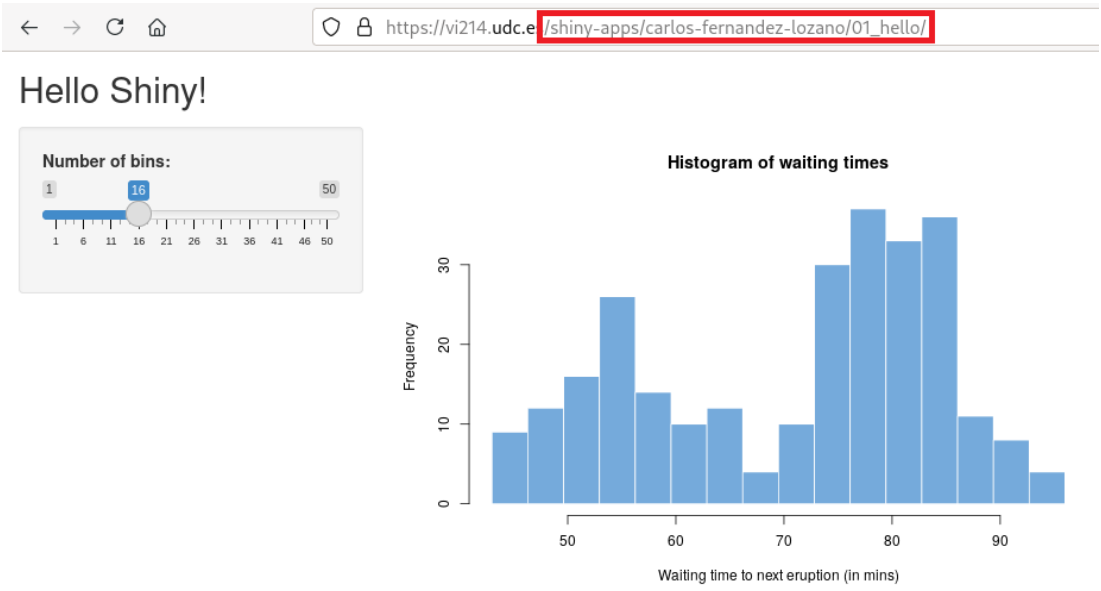


Figura 7.5: Ejemplo de aplicación Shiny accedida a través de Nginx como *proxy* inverso.

7.2 *Sprint* 2

7.2.1 H05 — Obtener datos de APIs públicas

Las APIs que se van a utilizar para obtener datos de investigadores son Publons y Google Scholar. Estos datos se obtienen con la ayuda de 3 *scripts*: 1 *script* en Shell para Publons y 2 *scripts* (en Shell y en Python) para Google Scholar. Una vez obtenidos, estos datos se almacenan en el fichero `_config.yml` del proyecto Jekyll, fichero donde se definen todas las variables que se pueden utilizar en los demás archivos que componen el proyecto.

El *script* usado en Publons (código 7.4), obtiene el número de revisiones y el número de ediciones del investigador, y lo realiza de la siguiente manera:

1. Se obtienen los datos de un investigador realizando una petición HTTP GET. Para esto, es necesaria la Uniform Resource Locator (URL) del perfil del investigador en Publons además de un *token*.

2. Se obtiene el dato *numberReviews* utilizando *jq*, un procesador de JSON para línea de comandos. Una vez obtenido, este se sustituye en la línea apropiada del fichero `_config.yml` utilizando otra herramienta de línea de comandos: *sed*.

3. Se repite el paso anterior, pero esta vez para el dato *numberEditor*.

4. Por último, se actualiza la fecha de última actualización de la obtención de datos de Publons a la fecha actual.

Es importante mencionar el uso de *sed* más *mv* en este código: a pesar de que *sed* posee la opción "-i", que permite realizar modificaciones directamente en los propios ficheros, está opción no cumple con la especificación Portable Operating System Interface (POSIX), lo cual resultaría en que un *script* que la utilice tendría resultados distintos dependiendo de si se ejecuta en un sistema Linux o en un sistema BSD/Mac. Esto se puede solucionar de diversas maneras, siendo una de ellas la que se ha realizado: guardar el resultado de la modificación del fichero en otro distinto, y luego sobreescribir el primero con los nuevos datos del segundo.

En cuanto a los *scripts* para Google Scholar, estos obtienen el número de citas y el *h-index* del investigador. En este caso, utilizamos el paquete *scholarly* de Python para descargar estos datos (código 7.5). Este *script*:

1. Obtiene el investigador del que se están buscando los datos.

2. Obtiene los datos del investigador a partir de una lista de secciones. Los dos datos que se quieren descargar de Google Scholar se encuentran en la sección "indices".

3. Se envían estos datos a la salida estándar.

Ahora, el *script* en Shell lo que hace es ejecutar el otro *script* (escrito en Python) y actualizar los datos que este devuelve en el fichero `_config.yml`, realizándolo de una forma parecida al *script* de Publons (código 7.6).

```
[...]

data="$(curl "${PUBLONS_URL}"                                    \
            --request  GET                                       \
            --header  "Authorization: Token ${TOKEN}"            \
            --header  "Content-Type:  application/json")"

[...]

numberReviews="$(echo "${data}" | jq '.reviews.pre.count')"

[...]

sed --expression
    's/^publons_number_reviews.*$/publons_number_reviews:
    '"${numberReviews}"'/g' _config.yml > _config.yml.bak \
&& mv _config.yml.bak _config.yml

[...]

numberEditor="$(echo "${data}" | jq
    '.handling_editor_records.count')"

[...]

sed --expression
    's/^publons_number_editor.*$/publons_number_editor:
    '"${numberEditor}"'/g' _config.yml > _config.yml.bak \
&& mv _config.yml.bak _config.yml

[...]

lastUpdate="$(date +"%Y-%m-%d")"
sed --expression 's/^publons_last_updated.*$/publons_last_updated:
    '"${lastUpdate}"'/g' _config.yml > _config.yml.bak \
&& mv _config.yml.bak _config.yml
```

Código 7.4: Shell *script* para la obtención de datos de investigador desde Publons.

```
#!/usr/bin/env python3

from scholarly import scholarly

AUTHOR                  = 'Fernandez-Lozano, Carlos'
SCHOLARLY_FILL_SECTIONS = ['indices']


search_query = scholarly.search_author(AUTHOR)
author       = next(search_query)
data         = scholarly.fill(author, sections =
    SCHOLARLY_FILL_SECTIONS)
print(f'''citedby: {data['citedby']}
hindex: {data['hindex']}''')
```

Código 7.5: Python *script* para la obtención de datos de investigador desde Google Scholar.

```
#!/bin/sh

data="$(python3 bin/.scholar.py)"

[...]

numberCitations="$(echo "${data}" | grep "^citedby.*$" | sed
    --expression "s/[^0-9]*//g")"

[...]

sed --expression 's/^scholar_number_cites.*$/scholar_number_cites:
    '"${numberCitations}"'/g' _config.yml > _config.yml.bak \
&& mv _config.yml.bak _config.yml

[...]

hindex="$(echo "${data}" | grep "^hindex.*$" | sed --expression
    "s/[^0-9]*//g")"

[...]

sed --expression 's/^scholar_hindex.*$/scholar_hindex:
    '"${hindex}"'/g' _config.yml > _config.yml.bak \
&& mv _config.yml.bak _config.yml

[...]

last_updated="$(date +"%Y-%m-%d")"
sed --expression 's/^scholar_last_updated.*$/scholar_last_updated:
    '"${last_updated}"'/g' _config.yml > _config.yml.bak \
&& mv _config.yml.bak _config.yml
```

Código 7.6: Shell *script* para la obtención de datos de investigador desde Google Scholar.

7.2.2 H06 — Flexibilizar la configuración de Nginx

Antes de comenzar con la explicación de la configuración de Nginx es necesario comentar que su imagen oficial de Docker tiene una función que permite utilizar variables de entorno para definir la configuración de Nginx, lo cual se implementa utilizando la herramienta *envsubst* que, dado un *string*, sustituye las variables entorno que se encuentren en él por sus respectivos valores. Esta imagen Docker ejecuta *envsubst* en ficheros con la extensión *template* para generar la configuración que Nginx leerá antes de arrancar.

Para realizar esta configuración se ha usado como base NGINXConfig, un sitio web que facilita la configuración de Nginx, aunque también se ha profundidado a través de la documentación oficial y múltiples *blogs* oficiales de Nginx, entre otros recursos.

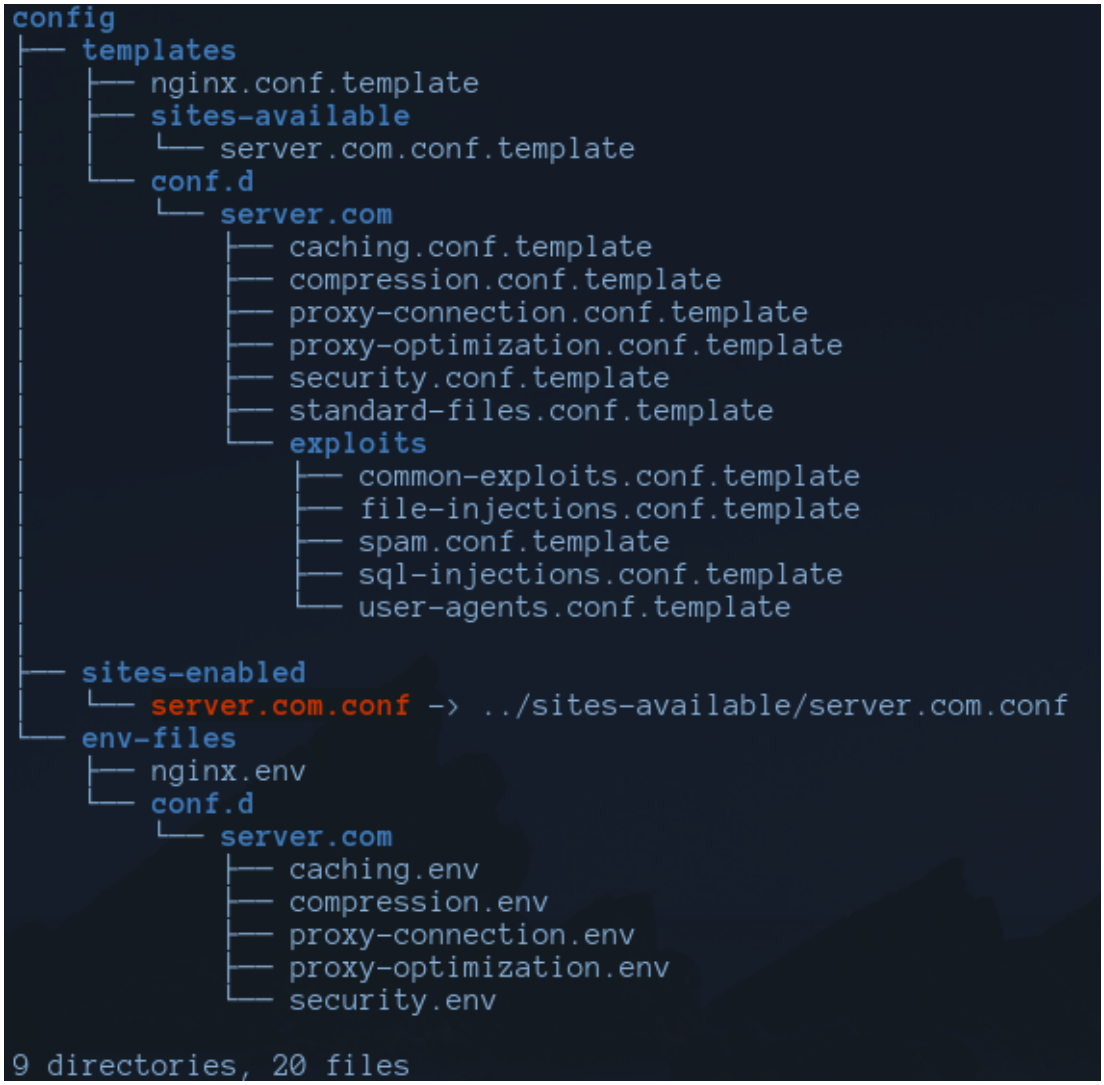


Figura 7.6: Árbol de ficheros de la configuración de Nginx.

El resultado de la configuración de Nginx es el árbol de directorios que puede verse en la figura 7.6, aunque se puede resumir en los siguientes puntos:

- El directorio `templates` contiene todos los ficheros que se utilizan para configurar Nginx. Este directorio se puede subdividir en:
  - El fichero `nginx.conf.template`, que contiene aquellas configuraciones que son comunes a todos los servidores virtuales de Nginx, como las que afectan al procesado de conexiones (bloque *events*) o a las conexiones seguras (por ejemplo, qué tipo de cifras y protocolos se deben usar), entre otras.
  - El directorio `sites-available` contiene un fichero con la configuración específica de cada servidor virtual gestionado por Nginx.
  - El subdirectorio `conf.d` contiene un directorio por cada servidor virtual que Nginx va a utilizar. En este caso, sólo se tiene uno (`server.com`) y contiene varios archivos que agrupan diferentes configuraciones por "temáticas". Destacar que el directorio `exploits` contiene configuraciones que intentar bloquear algunos *exploits* comunes. Estas configuraciones se han extraído de un proyecto de código abierto denominado nginx-proxy-manager.
- El directorio `sites-enabled` contiene ficheros que son enlaces simbólicos a aquellos servidores virtuales que se quieren utilizar. Comentar que estos enlaces simbólicos tienen que ser válidos dentro del contenedor Docker, y no en el *host*. Es por eso que, en la figura 7.6, el fichero `sites-enabled/server.com.conf` aparece en color rojo: el enlace simbólico no es válido en el *host* pero si lo será dentro del contenedor debido al cambio en la estructura de directorios.
- El directorio `env-files` contiene ficheros que declaran las variables en Shell cuyos valores serán sustituidos en las configuraciones de Nginx (directorio `templates`) una vez el contenedor comience su ejecución. De esta manera, si se quiere cambiar alguna configuración existente, únicamente sería necesario modificar el valor de la variable correspondiente en alguno de esos ficheros (ver figura 7.7).

Por último, hay que destacar que se ha intentado facilitar el futuro mantenimiento de esta configuración añadiendo comentarios para cada una de las directivas de Nginx, además de enlaces a la documentación oficial (ver figura 7.8).

7.2.3 H07 — Rotación de *logs* de los servicios desplegados

En la subsección A.1.2 del anexo se puede encontrar el porqué es apropiado implementar la rotación de los *logs* de nuestros servicios, además de qué pasos han sido necesarios para llevar a cabo esta tarea en el servidor de producción.

```
# https://nginx.org/en/docs/http/ngx_http_ssl_module.html#ssl_session_timeout
SSL_SESSION_TIMEOUT="4h"

# https://nginx.org/en/docs/http/ngx_http_ssl_module.html#ssl_session_cache
SSL_SESSION_CACHE="shared:SSL:20m"

# https://nginx.org/en/docs/http/ngx_http_ssl_module.html#ssl_session_tickets
SSL_SESSION_TICKETS="off"
```

Figura 7.7: Ejemplo de un fichero con variables entorno para configurar Nginx.

```
# NGINX will retain cached session parameters for 4 hours,
# which can improve performance because reusing cached parameters
# reduces the number of time-consuming handshakes.
# See: https://nginx.org/en/docs/http/ngx_http_ssl_module.html#ssl_session_timeout
ssl_session_timeout ${SSL_SESSION_TIMEOUT};

# Share the session cache among all worker processes speeds up later connections
# when the connection setup information is already known.
# See: https://nginx.org/en/docs/http/ngx_http_ssl_module.html#ssl_session_cache
ssl_session_cache ${SSL_SESSION_CACHE};

# Disable session tickets, which are an alternative to session cache
# that stores session information on the client side.
# See: https://nginx.org/en/docs/http/ngx_http_ssl_module.html#ssl_session_tickets
ssl_session_tickets ${SSL_SESSION_TICKETS};
```

Figura 7.8: Ejemplo de la configuración realizada en Nginx.

7.2.4 H08 — Securización del servidor

Nuevamente, en la sección A.2 del anexo se enumeran las diferentes medidas implementadas en el servidor que ayudan a mejorar su seguridad (junto con sus respectivos razonamientos).

7.2.5 H09 — Balancear la carga de los servicios con Docker Swarm

La configuración del balanceo de carga es realmente sencilla utilizando Docker Swarm: el modo Swarm de Docker tiene un componente DNS interno que asigna de forma automática a cada servicio del *swarm* una entrada DNS. El *swarm manager* usa un balanceador de carga interno para distribuir las peticiones entre los servicios dentro del *cluster*, basándose en el nombre DNS del servicio [50].

Es por esto que para balancear la carga de Shiny lo único que hay que realizar es cambiar el modo de despliegue a *replicated* y aumentar el número de réplicas del servicio, tal y como se puede observar en el código 7.7.

```
[...]
shiny:
  [...]
  deploy:
    mode: replicated
    replicas: 3
[...]
```

Código 7.7: Fragmento del fichero docker-compose.yml con configuración para el replicado de servicios.

Sin embargo, esto provoca que Nginx, que está configurado para ser un *proxy* inverso, falle en el arranque al intentar resolver el nombre *shiny* en su configuración (en concreto, enla directiva *proxy_pass*, directiva que indica a donde se debe reenviar el tráfico).

Esto se puede solucionar utilizando la herramienta *wait-for-it*, herramienta listada en la documentación oficial de Docker para controlar el orden del inicio y apagado de los diferentes servicios en una aplicación multicontenedor o *stack*. Este *script* en Bourne Again SHell (Bash) se ejecuta como comando de arranque del contenedor y se le indica un *hostname* y un puerto por el que esperar a que se tenga conexión y, una vez se tenga, se ejecutará el comando que se haya especificado. Se ha tenido que utilizar la opción *timeout* debido a que el tiempo máximo de espera definido de forma predeterminada es demasiado bajo (ver código 7.8).

```
[...]
nginx:
  [...]
  command: ["/wait-for-it.sh", "shiny:3838", "--timeout=90",
  "--", "nginx", "-g", "daemon off;"]
[...]
```

Código 7.8: Fragmento del fichero docker-compose.yml con configuración para controlar el orden de arranque.

Por último, se ha de añadir "/wait-for-it.sh" como opción dentro del *script* `docker-entrypoint.sh`, que se ejecuta en el arranque del contenedor (ver figura 7.9). Esto se debe a que el nuevo comando de arranque del contenedor es "wait-for-it.sh", por lo que, si no se añade, algunas funciones que tiene que realizar este *script* no se ejecutarían como deberían.

```
[...]
if [ "$1" = "nginx" -o "$1" = "nginx-debug" -o "$1" =
    "/wait-for-it.sh" ]; then
[...]
```

Código 7.9: Fragmento del fichero docker-entrypoint.sh

7.3 *Sprint* 3

7.3.1 H10 — Página web orientada al perfil corporativo

La plantilla de Jekyll elegida para la web corporativa ha sido minimal-mistakes. A continuación se listan algunas de sus funcionalidades:

- *Layouts responsive.*
- Soporte de tablas de contenido, imágenes cabecera, barras laterales, etc.
- Soporte de *tags* y categorías en noticias.
- Soporte de comentarios.
- ...

En las figuras 7.9, 7.10 y 7.11 se puede observar algunos ejemplos de la página web resultante.

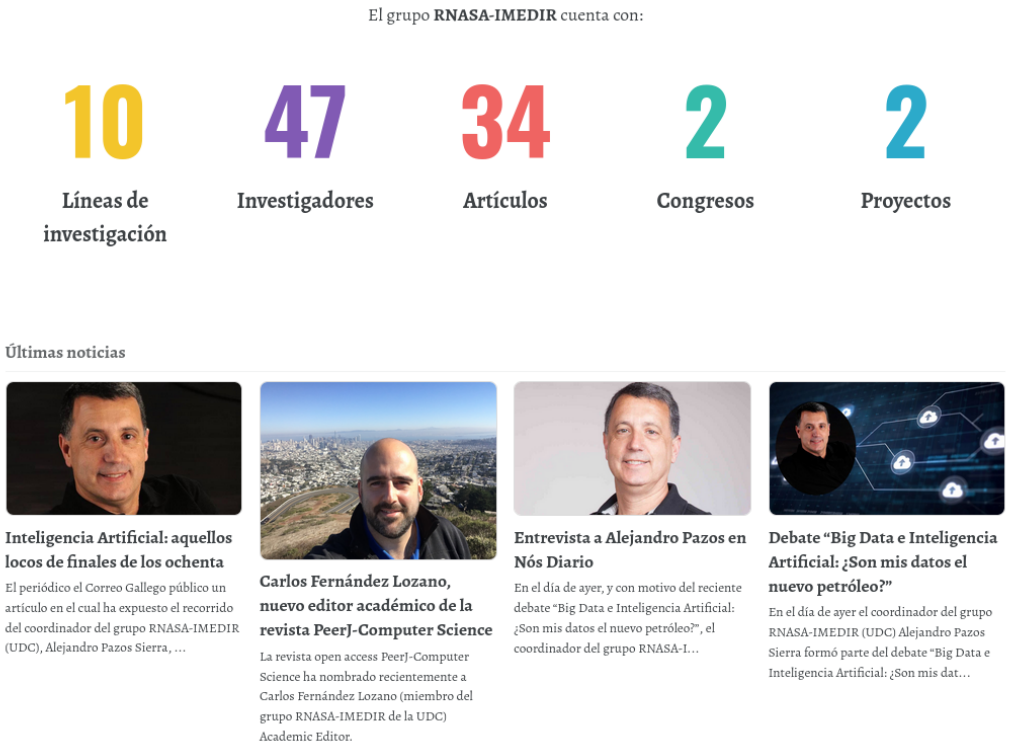


Figura 7.9: Fragmento de la página inicial del sitio web corporativo.

Figura 7.10: Fragmento de las líneas de investigación del sitio web corporativo.

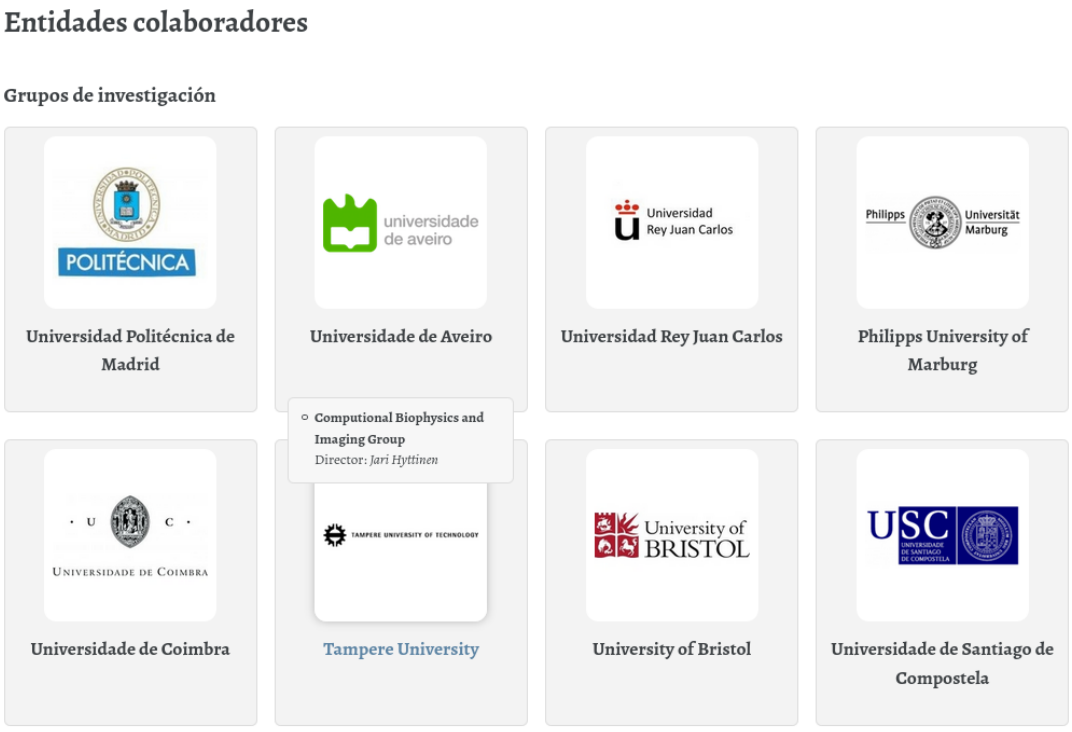


Figura 7.11: Fragmento de las entidades colaboradoras del sitio web corporativo.

7.3.2 H11 — Unificar la imagen corporativa del grupo de investigación

Una vez configurada la página web corporativa de la forma apropiada, se habrá terminado con la unificación de las distintas páginas web y servicios: en la figura 7.12 se puede ver como el perfil del investigador que se encuentra en la esquina superior izquierda resalta al mantener el ratón encima de él (debido a que es un enlace), pudiendo así acceder a su propia página web.

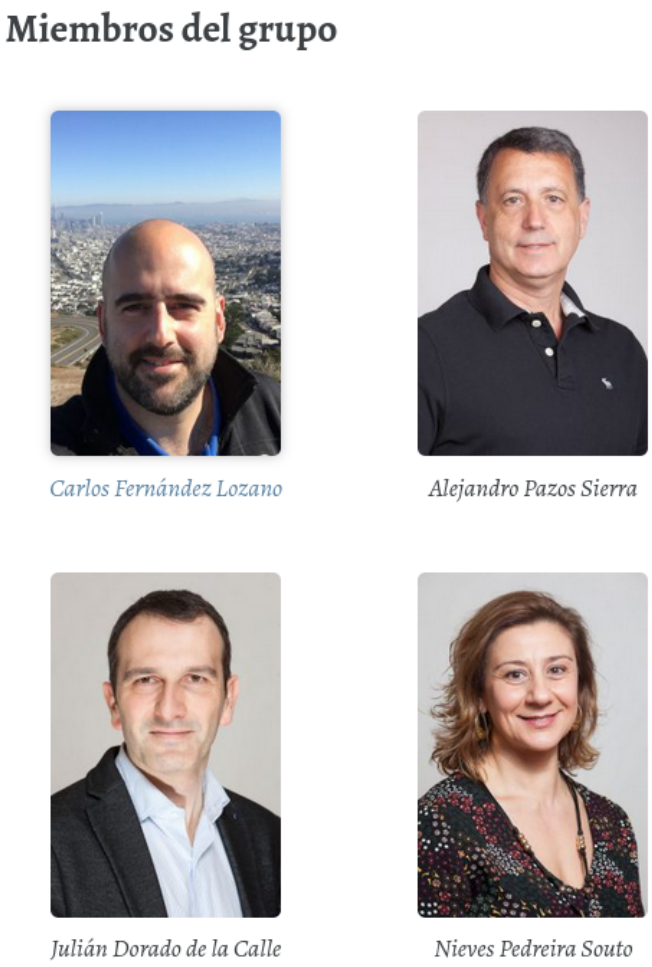


Figura 7.12: Acceso a los sitios web de los investigadores desde la página corporativa.

7.3.3 H12 — Configurar y desplegar fácilmente los sitios web en producción

Un problema que puede surgir con el paso del tiempo es que algún investigador desarrolle alguna aplicación en Shiny que necesite un paquete de R que no está incluido entre los que se pueden encontrar en la imagen de manera predeterminada. Para solucionar es-

te problema, se ha creado un *script* en Shell, denominado `install-packages.R`, que utiliza el argumento "*EXTRA_R_PACKAGES*" para instalar (en tiempo de construcción de la imagen) los paquetes que se le indiquen. Este *script* también utiliza los argumentos "*EXTRA_R_OLD_PACKAGES_NAMES*" y "*EXTRA_R_OLD_PACKAGES_VERSIONS*" en caso de que se necesite instalar alguna versión de un paquete que no sea la última.

Debido al mismo problema, la imagen personalizada de Shiny también utiliza otro *script* (`install-packages.sh`) para instalar, utilizando la herramienta *apt*, aquellos paquetes que sean necesarios (ver código 7.10).

```
#!/bin/sh

if [ -n "${EXTRA_PACKAGES}" ]; then
    apt update
    apt install --yes ${EXTRA_PACKAGES}
    rm --recursive --force /var/lib/apt/lists/*
fi
```

Código 7.10: Contenido del Shell *script* install-packages.sh.

En cuanto al servidor web, hasta este momento, en caso de querer desplegar Nginx en producción, habría que incluir en la imagen los ficheros generados por Jekyll. Esto supone tener instalado Jekyll y los *plugins* necesarios en alguna máquina, ejecutar Jekyll para obtener los ficheros que conforman los sitios web, y moverlos hasta el directorio apropiado en el servidor para que puedan ser copiados dentro de la imagen. Como esto no facilita la automatización, se ha modificado el fichero `Dockerfile` de Nginx para que este sea *multistage*, es decir, que tenga múltiples etapas de construcción utilizando una imagen base distinta en cada una de ellas [51].

Se utilizarán dos imágenes base (Jekyll y Nginx) en la creación de la imagen del servidor web. De esta manera solo se tendrá que copiar dentro de la imagen el código del proyecto Jekyll, y este código se convertirá en los ficheros del sitio web dentro del contenedor.

De la misma manera que con la imagen Shiny, en la imagen del servidor web se utilizará un *script* en Shell que instalará aquellas gemas que necesite Jekyll para generar las páginas web (ver código 7.11). Además, se modificará un segundo fichero de configuración de los proyectos Jekyll, que permitirá sobreescribir algunos parámetros de estos a unos valores que sean acordes al entorno de producción en el que se despliegan.

```
#!/bin/sh

if [ -n "${EXTRA_GEMS}" ]; then
    gem install ${EXTRA_GEMS}
fi
```

Código 7.11: Contenido del Shell *script* install-gem.sh.

Otro *script* en Shell, nombrado `build-sites.sh`, se ocupará de generar esos segundos ficheros de configuración y de compilar los proyectos Jekyll para obtener los sitios web (ver código 7.12).

```
[...]

find ${RESEARCHERS_SITES_DIR}/* -maxdepth 0 -type d -print | while
    read -r researcher_site_dir; do

    [...]

cp "${SITES_DIR}/config-prod.yml" "${researcher_site_config_prod}"
sed --expression
        "s/^\(baseurl:\).*/\1 \/miembros\/${researcher_name}/" \
    --expression
        "s%^\(\s\+source:\).*%\1 ${RESEARCHERS_BIB_FILES_DIR}%" \
    --expression
        "s/^\(\s\+bibliography:\).*/\1 ${researcher_name}.bib/" \
    "${researcher_site_config_prod}" >
    "${researcher_site_config_prod}.bak" \
&& mv "${researcher_site_config_prod}.bak"
    "${researcher_site_config_prod}"

jekyll build \
    --source "${researcher_site_dir}" \
    --config "${researcher_site_dir}/_config.yml,
    ${researcher_site_config_prod}" \
    --destination
        "${BUILDED_RESEARCHERS_SITES_DIR}/${researcher_name}"
done

[...]
```

Código 7.12: Fragmento del Shell *script* build-sites.sh.

También es necesario comentar que el proceso típico de trabajo con Docker sigue los siguientes pasos:

1. Construcción de una imagen.
2. *Push* de la nueva imagen a un repositorio.
3. *Pull* de la nueva imagen del repositorio y despliegue.

Esto es, la etapa de construcción de la imagen esta separada de su despliegue. Es por esto que se ha creado un nuevo *script* (`build-images.sh`) que se ocupa de la construcción

de imágenes, pudiendo así eliminar toda pista del proceso de construcción de los servicios del fichero `docker-compose.yml`, que ahora se dedicará exclusivamente a desplegar los *stacks* (ver código 7.13).

```
[...]

docker build \
    --build-arg EXTRA_GEMS="${EXTRA_GEMS}" \
    --build-arg NGINX_UID="${NGINX_UID}" \
    --build-arg NGINX_GID="${NGINX_GID}" \
    --tag "${RNASA_WEB_IMAGE_TAG}" \
    ./web/

[...]

docker build \
    --build-arg SHINY_UID="${SHINY_UID}" \
    --build-arg SHINY_GID="${SHINY_GID}" \
    --build-arg EXTRA_PACKAGES="${SHINY_EXTRA_PACKAGES}" \
    --build-arg EXTRA_R_PACKAGES="${SHINY_EXTRA_R_PACKAGES}" \
    --build-arg
    EXTRA_R_OLD_PACKAGES_NAMES="${SHINY_EXTRA_R_OLD_PACKAGES_NAMES}"
    \
    --build-arg
    EXTRA_R_OLD_PACKAGES_VERSIONS="${SHINY_EXTRA_R_OLD_PACKAGES_VERSIONS}"
    \
    --tag "${SHINY_CUSTOM_IMAGE_TAG}" \
    ./shiny/

[...]
```

Código 7.13: Fragmento del fichero build-images.sh.

Un problema que nos encontramos al realizar este cambio es que ahora se ha de buscar una alternativa para utilizar las variables entorno usadas para configurar Nginx (se habló de ellas en la subsección 7.2.2). También se ha utilizado un *script* en Shell para solucionar este problema: estos ficheros se copian dentro de la imagen y se exportan las variables que definen (es decir, se convierten en variables entorno) justamente antes de usar la herramienta *envsubst*, de la que también se habló en esa misma subsección pasada (ver código 7.14).

```
[...]

source_env () {

  [...]

  for file in $(find "${env_files_dir}" -follow -type f -name
    "*${env_files_suffix}" -print); do
    . "${file}"

    export $(grep --only-matching "^[^#][^=]*" "${file}")
  done
}

[...]
```

Código 7.14: Fragmento del fichero 20-envsubst-on-templates-custom.sh.

Por último, un diagrama representando la infraestructura desplegada y el flujo de información típico se puede observar en la figura 7.13.

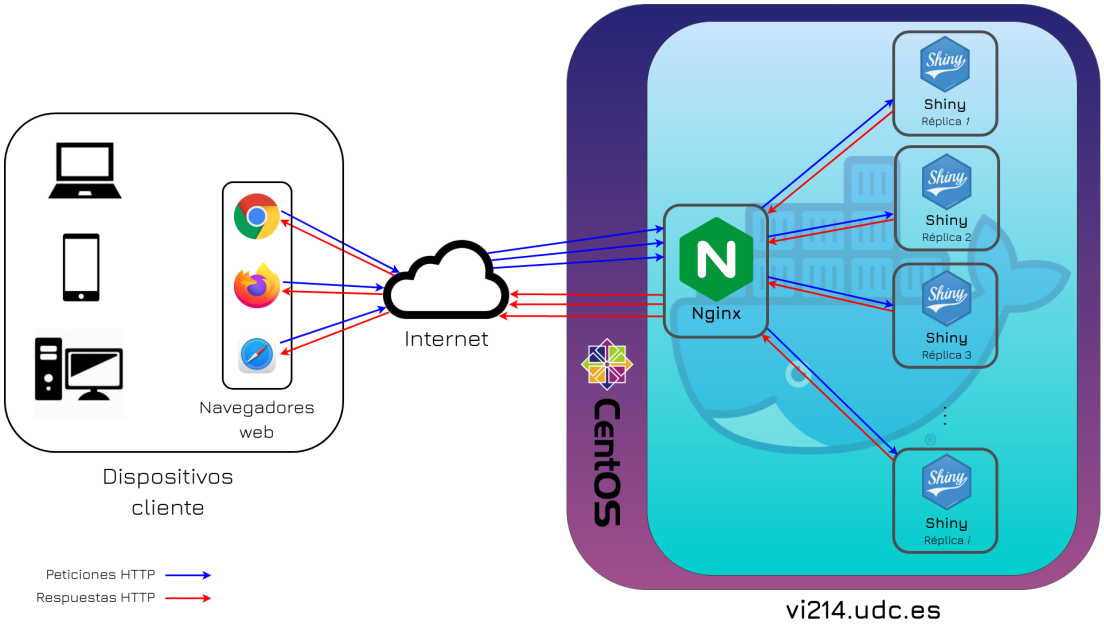


Figura 7.13: Flujo de información típico del sistema desplegado.

7.3.4 H13 — Mostrar la bibliografía de los investigadores en el sitio web corporativo

Acceder a toda la bibliografía desde la web corporativa supone tener un único fichero *bib* con esta bibliografía. Para que *jekyll-scholar* la utilice no hay más que dar los valores adecuados a las variables *source* y *bibliography*, de la misma manera que con la configuración de los investigadores (ver código 7.15).

El problema que surge es que juntar toda la bibliografía de los miembros del grupo supone que haya que controlar de alguna manera los posibles duplicados de las publicaciones, artículos, congresos, etc. Esto se puede solucionar utilizando la opción "--remove_duplicates" de las funciones que provee el *plugin jekyll-scholar*, pudiendo utilizar un campo de la bibliografía en el que basarse para controlar los duplicados. En el código 7.16 se puede apreciar este cambio (además del filtrado del tipo de bibliografía con la opción "--query").

```
[...]

cat ${RESEARCHERS_BIB_FILES_DIR}/*.bib >
    "${RESEARCHERS_BIB_FILES_DIR}/${merged_researchers_bibliography}"

[...]

sed --expression "s/^\(baseurl:\).*/\1 \//" \
    --expression
        "s%^\(\s\+source:\).*%\1 ${RESEARCHERS_BIB_FILES_DIR}%" \
    --expression "s/^\(\s\+bibliography:\).*/\1
    ${merged_researchers_bibliography}/" \
    "${rnasa_site_config_prod}" > "${rnasa_site_config_prod}.bak" \
&& mv "${rnasa_site_config_prod}.bak" "${rnasa_site_config_prod}"

[...]
```

Código 7.15: Parte del *script* Shell build-sites.sh donde se configura la bibliografía del sitio web corporativo en producción.

```
[...]

<li>
  <a href="{{ '/articulos/' | relative_url }}">
    <p id="index-data-articulos-count">{% bibliography_count
    --query @article --remove_duplicates doi %}</p>
    <strong class="index-data-count-caption">Artículos</strong>
  </a>
</li>
<li>
  <a href="{{ '/congresos/' | relative_url }}">
    <p id="index-data-congresos-count">{% bibliography_count
    --query @inproceedings --remove_duplicates doi %}</p>
    <strong class="index-data-count-caption">Congresos</strong>
  </a>
</li>
<li>
  <a href="{{ '/proyectos/' | relative_url }}">
    <p id="index-data-proyectos-count">{% bibliography_count
    --query @misc --remove_duplicates title %}</p>
    <strong class="index-data-count-caption">Proyectos</strong>
  </a>
</li>

[...]
```

Código 7.16: Fragmento del código de la página índice del sitio web corporativo.

Capítulo 8

Conclusiones

8.1 Resultados del proyecto

Tras casi 3 meses y medio desde el comienzo del proyecto, se dan por cumplidos los requisitos iniciales con los que empezó el trabajo, aunque también se han cumplido algunos de los requisitos que surgieron una vez el proyecto ya había comenzado su desarrollo. A continuación se listan las características más destacables del sistema resultante:

- Se ha creado y configurado una **página web de un investigador**. Desde esta página web se pueden acceder a los diferentes tipos de bibliografía que este liste (artículos, conferencias, proyectos de investigación, etc.), además de poder acceder a las distintas aplicaciones Shiny del mismo.
- Se ha creado y configurado una **página web corporativa** que representa el **grupo de investigación RNASA-IMEDIR**. Se ha replicado la estructura y los datos del sitio web que el grupo posee actualmente (presentación, miembros, líneas de investigación, colaboradores, etc.), añadiendo además nuevas funcionalidades, como la posibilidad de compartir noticias en algunas redes sociales o el listado en conjunto de la bibliografía del grupo.
- Estas dos páginas web han sido creadas utilizando **Jekyll**, una tecnología de generación de sitios web estáticos sencilla que facilitará su futuro mantenimiento.
- Se **obtienen datos de los investigadores a partir de APIs públicas** utilizando una serie de *scripts*.
- Se utiliza **Nginx** como **servidor web**, para servir los sitios web creados, y como ***proxy* inverso**, para acceder a las **aplicaciones Shiny** creadas por los investigadores. La configuración de Nginx se ha flexibilizado para poder realizar grandes cambios en ella sin tener que tocar los propios ficheros fuente.

- También se ha **flexibilizado el entorno en el que se despliegan aplicaciones Shiny**, pudiendo instalar, de una manera sencilla, nuevos paquetes de R que puedan ser necesarios en la ejecución de futuras aplicaciones Shiny.
- Estos servicios se despliegan mediante un *stack* utilizando **Docker, Docker-Compose y Docker Swarm**, obteniendo así el balanceo de carga necesario para gestionar las diferentes aplicaciones Shiny.
- Todo este sistema se ha desplegado en un **servidor de producción CentOS**. Además del propio despliegue de la aplicación, también se ha configurado la **rotación de los *logs*** de los diferentes servicios involucrados y se ha **aumentado la seguridad del servidor** implementando diversas medidas como, por ejemplo, múltiples factores de autenticación.

8.2 Posibles mejoras

- En diciembre de 2020, Red Hat anunció que adelantaba la fecha de *End-of-life* desde 2029 al 31/12/2021 para CentOS, discontinuándolo y sustituyéndolo por CentOs Stream, que será usado como *upstream* para Red Hat Enterprise Linux (RHEL), es decir, como distribución para probar aquellas actualizaciones que más adelante se incluirán en la distribución de pago de Red Hat [52].

  Como reacción a este anuncio, están apareciendo nuevas distribuciones como Rocky Linux o AlmaLinux, que intentan sustituir el espacio que ha dejado libre CentOS.

  Sería recomendable estudiar cual sería la mejor dirección a tomar de cara al futuro del servidor.
- Debido a los riesgos de seguridad que supone que Docker se ejecute con privilegios de *root*, sería recomendable estar atento a los avances en el desarrollo de Podman para poder reemplazar Docker por una alternativa más segura. En caso de querer mantener Docker, se podría utilizar la capacidad *userns-remap* para evitar posibles ataques de escalado de privilegios [53].
- Se podrían utilizar las directivas del módulo *ngx_http_limit_req* de Nginx en caso de querer implementar una medida de protección ante ataques *Distributed Denial-of-Service (DDoS)* [54]. Muy recomendable hacer pruebas antes de desplegar esta nueva configuración en producción.
- La integración de módulos dentro de la configuración de Nginx podría ser interesante. Por ejemplo, el módulo *ngx_brotli* podría aportar unos niveles de compresión mejores que los que se tienen actualmente utilizando *gzip*.

- Para realizar un despliegue, se necesitan 4 conjuntos de datos:
  - Página web RNASA-IMEDIR.
  - Páginas web de los investigadores (en conjunto).
  - Bibliografía de los investigadores.
  - Aplicaciones Shiny de los investigadores.

  Actualmente, solo la web corporativa se puede obtener enteramente de un repositorio.

  Si se pudieran obtener todos de repositorios, se podría añadir herramientas de *Continuous Integration (CI)*/*Continuous Delivery (CD)* al sistema creado para así automatizar todo el proceso de testeo, construcción y despliegue de imágenes Docker.

8.3 Relación con las competencias de la titulación

Gran parte de las habilidades necesarias para efectuar correctamente este proyecto se han adquirido en diversas asignaturas a lo largo del grado:

- **Administración de Infraestructuras y Sistemas Informáticos**. Esta asignatura fue vital a la hora de manejar Docker para el despliegue de los diferentes servicios que componen este proyecto.
- **Administración de Sistemas Operativos**. Las distintas tareas administración llevada a cabo en el servidor CentOS, como la rotación de *logs* o la configuración de los Pluggable Authentication Module (PAM), se han de agradecer a haber cursado esta materia.
- **Programación Integrativa**. Los conocimientos aprendidos en esta asignatura ayudaron en la creación de varios códigos en lenguajes de *scripting*.
- **Interfaces Hombre Máquina**. La introducción en el desarrollo web, producida al cursar esta asignatura, fue de gran ayuda para diseñar los sitios de los investigadores y del grupo RNASA-IMEDIR.
- **Legislación y Seguridad Informática**. Parte de las medidas de seguridad implementadas en el servidor (como el cortafuegos) se deben a esta materia.
- **Redes**. Importante en diversos puntos en los que se requería configurar la comunicación entre las distintas partes del sistema creado.

Apéndices

Apéndice A

Despliegue en CentOS

A.1 Despliegue del proyecto

En esta sección se muestran los pasos que se deben realizar para desplegar el sistema creado. A continuación se listan los repositorios donde se puede encontrar los códigos fuentes y las configuraciones de los servicios a desplegar, además de la configuración de Docker:

- Proyecto Jekyll del sitio web del grupo RNASA-IMEDIR: https://github.com/rodrigo-vb/rnasa-jekyll.
- Proyecto Jekyll del sitio web de investigador: https://github.com/rodrigo-vb/rnasa-researcher-jekyll.
- Configuración de Docker, Nginx, Shiny y su rotación de logs: https://github.com/rodrigo-vb/rnasa-docker.

A.1.1 Despliegue en Docker Swarm

Para poder llevar a cabo el despliegue, primero se necesita obtener el repositorio de Git que contiene toda la configuración:

```
1 $ git clone https://github.com/rodrigo-vb/rnasa-docker
```

Ahora se descargan, en los directorios correspondientes, los repositorios que contienen los códigos que componen tanto la página web del grupo de investigación como la página web de un investigador:

```
$ mkdir --parent ~/rnasa-docker/web/jekyll/sites/researchers
$ git clone https://github.com/rodrigo-vb/rnasa-jekyll
    ~/rnasa-docker/web/jekyll/sites/rnasa
$ git clone https://github.com/rodrigo-vb/rnasa-researcher-jekyll
    carlos-fernandez-lozano
    ~/rnasa-docker/web/jekyll/sites/researchers/carlos-fernandez-lozano
```

Es recomendable crear un enlace simbólico para acceder con facilidad a estos repositorios:

```
$ ln --symbolic ~/rnasa-docker/web/jekyll/sites ~/web-sites
```

También se han de enviar al servidor los ficheros *bib* (que contienen la bibliografía de los distintos investigadores del grupo) y las aplicaciones en Shiny desarrolladas también por los investigadores. Esto puede realizarse utilizando, por ejemplo, la herramienta de línea de comandos *scp*. Una vez hecho esto, simplemente hay que situarlos en los directorios correspondientes. Suponiendo que se tienen ficheros en formato *zip*, los comandos a ejecutar serían los siguientes:

```
$ mkdir ~/rnasa-docker/web/jekyll/data
$ unzip ~/researchers-bibs.zip -d ~/rnasa-docker/web/jekyll/data/researchers-bibs
$ unzip ~/shiny-apps.zip -d ~/rnasa-docker/shiny/rnasa-shiny-apps
```

Como último paso de preparación del despliegue, se deben crear los directorios donde los servicios almacenarán sus *logs*. Esto se debe a que, si no se encuentran creados en el momento en el que los servicios se despliegan, Docker los creará automáticamente pero con permisos de *root* (dado que son utilizados como *bind mounts*), lo cual hará que los servicios fallen al no tener los permisos necesarios para interactuar con esos directorios.

```
$ mkdir --parent ~/rnasa-docker/logs/nginx ~/rnasa-docker/logs/shiny
```

Finalmente, para desplegar el sistema no hay más que construir las imágenes de los contenedores utilizando el script creado para ello, iniciar el *swarm* y desplegar el *stack* utilizando el fichero docker-compose.yml:

```
$ cd ~/rnasa-docker
$ sh ./build-images.sh
$ docker swarm init
$ docker stack deploy --compose-file ./docker-compose.yml rnasa
```

Si en algún momento se necesita parar la ejecución del *stack*, no hay más que ejecutar:

```
$ docker stack rm rnasa
```

A.1.2 Rotación de logs

Los dos servicios que componen el *stack* están expuestos a Internet y *logean* cada una de las conexiones que reciben. Es por ello que es importante rotar sus *logs*: estos pueden llegar a ocupar todo el espacio disponible del sistema, dejándolo inutilizado. Para esta tarea se van a utilizar la herramientas *logrotate*, que se ocupa de automatizar la rotación, compresión y eliminación de los *logs* (entre otras tareas), y *cron*, que se ocupará de que la rotación de los *logs* se realice de forma periódica.

Se han planteado dos maneras de realizar esta tarea:

- Rotar los *logs* desde dentro de los contenedores.
- Rotar los *logs* desde fuera de los contenedores.

Se ha optado por la segunda opción, ya que la primera implica que cada una de las imágenes de los contenedores tendría que contener las herramientas *logrotate* y *cron*, junto con sus respectivas configuraciones. En cambio, con la segunda opción podemos tener todo el *software* y configuraciones en el *host* de una manera más unificada.

En los códigos A.3 y A.4 se puede ver la configuración de la rotación de los *logs* para los dos servicios. Las dos configuraciones rotan los *logs* de forma diaria y los mantienen durante 30 días antes de comenzar a borrar los más antiguos. La única diferencia notable entre las dos configuraciones es que a Nginx se le envía (con la directiva *postrotate*) la señal "USR1" después de rotar los *logs* para que este comience a utilizar unos ficheros nuevos para *logear* las conexiones, mientras que con Shiny los *logs* se tienen que copiar y, luego, truncar el fichero original, ya que no proporciona la misma funcionalidad que Nginx.

Para configurar la rotación de *logs* en el sistema primero se deben crear dos usuarios, uno para cada servicio. Es importante mencionar que estos usuarios deben utilizar UIDs y GIDs que se sepa que no están asignados a usuarios dentro de los contenedores de los servicios. Esto nos evitará posibles problemas con los permisos si, por un casual, el UID o el GID del usuario que gestiona los *logs* en el *host* se corresponde con un usuario dentro del contenedor que no es quién maneje los *logs*.

```
$ sudo groupadd --gid 30000 --system rnasa-nginx
$ sudo useradd --uid 30000 --gid rnasa-nginx --system rnasa-nginx
$ sudo groupadd --gid 40000 --system rnasa-shiny
$ sudo useradd --uid 40000 --gid rnasa-shiny --system rnasa-shiny
```

A continuación, se crean los directorios donde se almacenarán los *logs* y se les dan los permisos adecuados:

```
$ mkdir /var/log/rnasa-nginx /var/log/rnasa-shiny
$ chown --recursive rnasa-nginx:rnasa-nginx /var/log/rnasa-nginx
$ chown --recursive rnasa-shiny:rnasa-shiny /var/log/rnasa-shiny
$ chmod --recursive u=rwX,g=rX,o= /var/log/rnasa-nginx /var/log/rnasa-shiny
```

Ahora se copian las configuraciones de las rotaciones de *logs* y de *cron* a sus respectivos lugares, se les dan los permisos correspondientes y se inicia *cron* (tanto en el momento como en el arranque del sistema):

```
$ cp ~/rnasa-docker/log-rotation/nginx.logrotate /etc/logrotate.d/rnasa-nginx
$ cp ~/rnasa-docker/log-rotation/shiny.logrotate /etc/logrotate.d/rnasa-shiny
$ cp ~/rnasa-docker/log-rotation/crontab /etc/crontab
$ chown root:root /etc/logrotate.d/rnasa-nginx /etc/logrotate.d/rnasa-shiny /etc/crontab
$ chmod u=rw,g=r,o=r /etc/logrotate.d/rnasa-nginx /etc/logrotate.d/rnasa-shiny
    /etc/crontab
$ systemctl enable --now cron
```

El único de detalle que queda es modificar la configuración del *stack*:

- Se modifican, en el fichero `build-images.sh`, los valores de las variables *_UID y *_GID tanto de Nginx como de Shiny (código A.1).
- Se configura, el fichero `docker-compose.yml`, los *bind mounts* entre los directorios de los *logs* en el *host* y en los contenedores (código A.2)

Para terminar no hay más que reconstruir las imágenes de los contenedores y desplegar el *stack* (de la misma manera que en la sección anterior) para que se puedan ver los frutos de esta configuración.

```
[...]

readonly NGINX_UID="30000"
readonly NGINX_GID="30000"

[...]

readonly SHINY_UID="40000"
readonly SHINY_GID="40000"

[...]
```

Código A.1: Fragmento del fichero build-images.sh con configuración para la rotación de *logs*.

```
[...]
nginx:
  [...]
  volumes:
    - /var/log/rnasa-nginx/:/var/log/nginx/:rw
[...]
shiny:
  [...]
  volumes:
    - /var/log/rnasa-shiny/:/var/log/shiny-server/:rw
[...]
```

Código A.2: Fragmento del fichero docker-compose.yml con configuración para la rotación de *logs*.

```
"/var/log/rnasa-nginx/*.access.log"
    "/var/log/rnasa-nginx/*.error.log" {
    # Log files are rotated every day.
    daily

    # Log files are rotated 30 times before being removed.
    rotate 30

    # Don't issue an error message when log files are missing.
    missingok

    # Run postrotate script one time per rotation instead of one
    time per log rotated.
    sharedscripts

    # Compress log files.
    compress

    # Wait until the next rotation cycle to compress log files.
    delaycompress

    # Archive log files by adding a date as extension.
    dateext

    # Use the day before as extension.
    dateyesterday

    # Run this script after rotating log files.
    # See:
    #   -
    https://www.nginx.com/resources/wiki/start/topics/examples/logrotation
    #   - https://nginx.org/en/docs/control.html (Rotating
    Log-files section).
    postrotate
        service="$(docker ps | grep rnasa_nginx)"
        container_name="$(echo "${service##* }")"
        docker kill -s USR1 "${container_name}" >/dev/null 2>&1
    endscript
}
```

Código A.3: Configuración de la rotación de logs de Nginx.

```
"/var/log/rnasa-shiny/*.log" {
    # Log files are rotated every day.
    daily

    # Log files are rotated 30 times before being removed.
    rotate 30

    # Truncate the original log file to zero size in place
    # after creating a copy.
    copytruncate

    # Don't issue an error message when log files are missing.
    missingok

    # Compress log files.
    compress

    # Wait until the next rotation cycle to compress log files.
    delaycompress

    # Archive log files by adding a date as extension.
    dateext

    # Use the day before as extension.
    dateyesterday
}
```

Código A.4: Configuración de la rotación de logs de Shiny.

```
[...]
# Example of job definition:
# .---------------- minute (0 - 59)
# |   .------------- hour (0 - 23)
# |   |   .---------- day of month (1 - 31)
# |   |   |   .------- month (1 - 12) OR jan,feb,mar,apr ...
# |   |   |   |   .---- day of week (0 - 6) (Sunday=0 or 7) OR
    sun,mon,tue,wed,thu,fri,sat
# |   |   |   |   |
# *   *   *   *   *  user-name  command to be executed
  00  00  *   *   *  root       logrotate
    /etc/logrotate.d/rnasa-nginx
  00  00  *   *   *  root       logrotate
    /etc/logrotate.d/rnasa-shiny
```

Código A.5: Configuración de la automatización de la rotación de logs con *cron*.

A.2 Securización del servidor

En esta sección se habla acerca de los pasos a realizar en caso de que se quiera mejorar la seguridad del servidor que corre el sistema desplegado. Se comenta como configurar un firewall y un software de prevención de intrusiones, además de la implementación de políticas de contraseñas, de múltiples factores de autenticación y, finalmente, también se trata la configuración del demonio SSH. En el siguiente enlace se puede encontrar el repositorio que contiene todos los ficheros de los que se habla en este apartado: https://github.com/rodrigo-vb/rnasa-security-configs.

A.2.1 Protegiendo el servidor de conexiones externas no deseadas

UFW como firewall

Como primera barrera entre Internet y el servidor se utilizará Uncomplicated Firewall (UFW). Este firewall, que no es más que un *frontend wrapper* de *iptables*, ha sido desarrollado para ser fácil de usar y encaja bien en nuestro caso de uso ya que solamente se necesita controlar el acceso a un pequeño número de puertos de nuestro servidor.

Para instalar este firewall primero tenemos que instalar `epel-release`, un paquete que permite descargar software del repositorio Extra Packages for Enterprise Linux (EPEL), que contiene paquetes que no se encuentra en los repositorios oficiales de Red Hat.

```
$ sudo yum install --assumeyes epel-release
$ sudo yum install --assumeyes ufw
```

Antes de comenzar con la configuración de UFW, hay que destacar que con esta herramienta se pueden establecer reglas de filtrado a partir de perfiles de aplicaciones, conteniendo cada uno de estos el nombre, una descripción y el puerto o puertos que utiliza la aplicación. Junto a la instalación de UFW también se descargan, de manera automática, una gran cantidad de perfiles que definen los servicios más utilizados (relacionados con servidores web, servidores de correo, de DNS, etc.), y se pueden encontrar en el directorio `/etc/ufw/applications.d`.

Dado que en el servidor sólo se tendrán dos servicios accesibles desde Internet (un servidor SSH y un servidor web/*proxy* inverso), únicamente se necesita permitir el acceso a los puertos donde estos servicios se encuentran y denegar todo lo demás. Para ello, se permite el acceso exclusivamente a la aplicación "SSH" y a la aplicación "WWW Full" (esto es, servidores HTTP y HTTPS) y se deniega todo el resto del tráfico. Al contrario que el acceso a los servidores web, el acceso mediante SSH estará limitado a 5 conexiones cada 30 segundos por dirección IP (todas las demás conexiones que se realizan dentro de ese rango de tiempo serán denegadas), añadiendo así una capa de protección contra ataques de fuerza bruta. Todo esto se realiza con los siguientes 3 comandos:

```
$ sudo ufw limit "SSH"
$ sudo ufw allow "WWW Full"
$ sudo ufw default deny incoming
```

El tráfico de salida se permite libremente y el tráfico *routed* (*forwarded* o encaminado) se deniega:

```
$ sudo ufw default allow outgoing
$ sudo ufw default deny routed
```

Para terminar, se ejecutan los dos siguientes comandos para que UFW arranque junto con el sistema y para activarlo:

```
$ sudo systemctl enable ufw
$ sudo ufw enable
```

Con el comando "$ sudo ufw status verbose" se puede observar el estado detallado de UFW (figura A.1) después de aplicar las reglas enumeradas en este apartado.

```
Status: active
Logging: off
Default: allow (incoming), allow (outgoing), disabled (routed)
New profiles: skip

To                         Action      From
--                         ------      ----
22/tcp (SSH)               LIMIT IN    Anywhere
80,443/tcp (WWW Full)      ALLOW IN    Anywhere
22/tcp (SSH (v6))          LIMIT IN    Anywhere (v6)
80,443/tcp (WWW Full (v6)) ALLOW IN    Anywhere (v6)
```

Figura A.1: Estado detallado de UFW.

Fail2ban como software de prevención de intrusiones

A pesar de que UFW ofrece limitación del ratio de conexiones, esto puede no ser lo suficientemente potente. Para solucionar este problema se puede utilizar *fail2ban*, una aplicación escrita en Python que monitoriza los *logs* de otras aplicaciones y añade (o elimina) reglas al firewall que se le configure. Este software es principalmente utilizado con servidores SSH, aunque realmente puede utilizarse con cualquier aplicación que almacene direcciones IP en sus *logs*.

Para instalar fail2ban es necesario ejecutar el siguiente comando (con el repositorio EPEL previamente instalado):

```
1 $ sudo yum install --assumeyes fail2ban
```

Para configurar fail2ban con demonio SSH se ha de añadir la configuración necesaria (que se puede ver en el código A.6) al directorio `/etc/fail2ban/jail.d`. De forma resumida, esta configuración hace que fail2ban:

- Monitorice los *logs* que el demonio SSH almacena.
- Banee (y desbanee) utilizando UFW.
- Banee, durante 15 minutos, toda dirección IP que realice más de 5 conexiones en un intervalo de tiempo de 30 minutos.
- Doble el tiempo de baneo de manera progresiva para aquellas direcciones IP que continúen intentando acceder al servidor.

Una vez hecho esto, se puede activar fail2ban (tanto en el momento como en el arranque del sistema) ejecutando:

```
1 $ sudo systemctl enable --now fail2ban
```

Por último comentar que, dado que ahora es fail2ban quién se dedica a limitar las conexiones y no UFW, lo ideal es eliminar el limitado de conexiones al servidor SSH que se introdujo antes (con el comando "$ sudo ufw limit SSH"). Esto se realiza ejecutando:

```
1 $ sudo ufw allow SSH
```

```
[sshd]

# Enable fail2ban.
enabled = true

# Specify filter (used to identify failed authentication attempts
    in log files
# and to extract the host IP address).
filter = sshd

# Specify action to apply on ban (commands execute on ban, on
    unban, etc.).
banaction = ufw[application=SSH]

# Effective ban duration
bantime = 15m

# If enabled, bantime will increase with each ban when the IP
    address is known.
# See: https://github.com/fail2ban/fail2ban/pull/1460
# Only in fail2ban v0.11+
bantime.increment = true

# Default factor (correspond to twice increase).
# Only in fail2ban v0.11+
bantime.factor = 1

# Avoid endless growth of bantime.
# Only in fail2ban v0.11+
bantime.maxtime = 1w

# Time interval before the current time where failures will count
    towards a ban.
findtime = 30m

# Number of failures that have to occur in the last findtime
    interval to ban
# the IP address.
maxretry = 5

# Indicates if the banning of own IP addresses should be prevented.
ignoreself = true
```

Código A.6: Configuración de fail2ban para el servidor SSH.

A.2.2 Política de contraseñas

Las políticas de contraseñas son reglas diseñadas para que los usuarios empleen contraseñas consideradas seguras. En este apartado se han seguido algunas de las recomendaciones del National Institute of Standards and Technology (NIST) [55] a la hora de implementar una política de contraseñas en el servidor, siendo algunas de ellas:

- **Requerir un tamaño mínimo de contraseña**. Contraseñas demasiado cortas aumentan la probabilidad de ser descubiertas en ataques de fuerza bruta o de diccionarios.
- **Evitar el reuso de contraseñas**.
- **No imponer el uso contraseñas complejas** (por ejemplo, combinar distintos tipos de caracteres). Contraseñas complejas introducen el nuevo riesgo de que son más complicadas de memorizar, por lo que es más probable que sean escritas o almacenadas de forma insegura.
- **No imponer el cambio periódico de contraseñas**. Requerir cambios frecuentes de contraseñas produce que los usuarios busquen maneras de que sus contraseñas sean variantes de antiguas (por ejemplo, "password", "password1", "password2", etc.), debilitando el propio objetivo que supone el forzar un cambio de contraseñas.
- **Uso de autenticación de múltiples factores**. Añadir un nuevo factor de autenticación añade una nueva capa de seguridad.

Implementar estas políticas requiere utilizar los PAM. Estos módulos, utilizados por un gran número de sistemas tipo UNIX, son un mecanismo que provee de una API generalizada a las aplicaciones para utilizar servicios relacionados con la autenticación, evitando así que cada aplicación los implemente a su propia manera.

El requerimiento de un tamaño mínimo de contraseñas y el control de su reuso se han realizado modificando el fichero `/etc/pam.d/system-auth` (se puede ver la parte relevante en el código A.7). Se obliga a que las contraseñas tengan un mínimo de 16 caracteres y se comprueba que no se repitan con alguna de las 20 anteriores.

```
[...]
password    requisite     pam_pwquality.so [...] minlen=16
password    requisite     pam_pwhistory.so  use_authok remember=20
  retry=3
[...]
```

Código A.7: Configuración del PAM system-auth para requerir un tamaño mínimo y evitar el reuso de contraseñas.

Por otro lado, el requerimiento de la autenticación de múltiples factores de autenticación lo llevaremos a cabo mediante el PAM Google Authenticator, que permite el uso de One-Time Passwords (OTPs). Para ello, necesitamos instalar los paquetes `google-authenticator` (que provee del PAM para poder autenticarnos con estos códigos y de una herramienta para generar el secreto a partir del cual se crearán nuestros códigos) y `qrencode` (permitirá crear un código Quick Response (QR) que se facilitará la obtención del secreto). Por lo tanto, habiendo instalado previamente el repositorio EPEL, se ha de ejecutar el siguiente comando:

```
1 $ sudo yum install --asummeyes google-authenticator qrencode
```

Una vez hecho, tendremos que modificar el fichero `/etc/pam.d/sshd` para activar el segundo factor de autenticación utilizando el nuevo PAM instalado (se puede ver la parte relevante del fichero modificado en el código A.8). Mencionar que la opción "nullok" del PAM permite que aquellos usuarios que no tengan configurado el segundo factor de autenticación se puedan autenticar, permitiendo que esos usuarios no se queden fuera del sistema una vez se active esta configuraci'on.

```
[...]
auth       substack     password-auth
auth       required     pam_google_authenticator.so nullok
[...]
```

Código A.8: Configuración del PAM sshd para la autenticación por OTP.

La segunda y última modificación a realizar será en la configuración del demonio SSH en el fichero `/etc/ssh/sshd_config`, donde tendremos que asegurarnos de que las opciones *ChallengeResponseAuthentication* y *UsePam* estén activadas, las cuales nos permitirán autenticarnos con nuestros OTPs. En el código A.9 se puede ver como debería resultar esa parte de la configuración.

```
[...]
## If enabled, allow challenge-response authentication.
ChallengeResponseAuthentication yes

## If enabled enable PAM authentication using
  ChallengeResponseAuthentication
## and PasswordAuthentication.
UsePAM yes
[...]
```

Código A.9: Configuración del demonio SSH para la activación de la autenticación por OTP.

Una vez hechos todos los cambios, comprobamos que la configuración de SSH no contiene errores y, si es así, reiniciamos el demonio para que la nueva configuración tome efecto:

```
$ sudo sshd -t
$ sudo systemctl restart sshd
```

Para poder autenticarse utilizando el nuevo factor tendremos que generar el secreto a partir del que se crearán los OTPs. Para ello se utiliza la herramienta que hemos instalado con anterioridad (`google-authenticator`) la cual permite elegir si la generación de los OTPs tendrá en cuenta la fecha y hora (Time-based One-Time Password (TOTP)) o no (HMAC-based One-Time Password (HOTP)). Si se ha instalado también el paquete `qrencode`, se nos proporcionará un código QR (ver figura A.2) que podrá ser escaneado con nuestra aplicación de verificación en 2 pasos preferida (a pesar del nombre del módulo PAM, este no restringe a únicamente poder utilizar *Google Authenticator* como este tipo de aplicación).

Figura A.2: Generación del secreto para la autenticación en 2 pasos.

Destacar que, si el tipo de código OTP que se elige está basado en tiempo (es decir, TOTP) es muy importante que la fecha y hora del sistema sean correctas, o de lo contrario no podremos conectarnos de forma exitosa.

Terminada la generación del secreto, cuando se intente conectar al servidor, se nos pedirá un código de verificación después de la contraseña para que la autenticación sea exitosa (figura A.3).

Por último, las dos políticas restantes (no imposición de contraseñas complejas y su cambio periódico) ya están implementadas de forma predeterminada, por lo que no es necesario

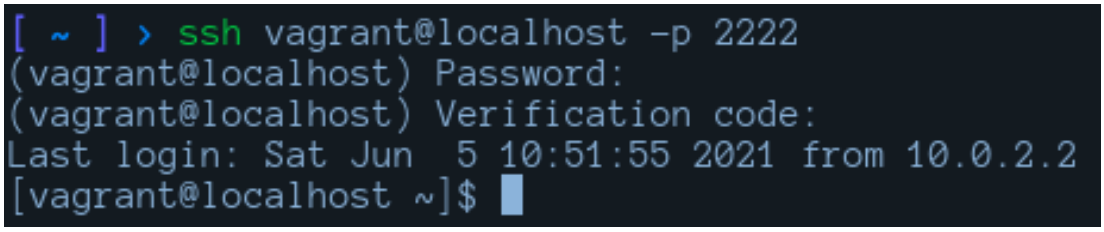


Figura A.3: Autenticación en 2 pasos (contraseña, código OTP) con el servidor SSH.

realizar ningún cambio en el sistema.

A.2.3 Configuración del demonio SSH

Para finalizar la securización del servidor remoto, se ha modificado la configuración del demonio SSH para cambiar la autenticación por contraseña por autenticación de clave pública, entre otros cambios. En el código A.11 se pueden ver las partes del fichero de configuración que se han considerado más relevantes, pero a continuación se explican con algo más de detalle:

- Se cambia el puerto donde SSH escucha. A pesar de que este cambio no aporta seguridad de por sí, ayuda a reducir el *log spam* producido por herramientas automatizadas que tratan de conectarse al puerto predeterminado de SSH.
- Se deniega el login del *root* a través de SSH, se deniega la autenticación mediante contraseña, se activa la autenticación desafío-respuesta (mediante los PAM) y se especifica que la autenticación seguirá dos pasos: el primero será autenticación por clave pública, y el segundo será interactivo (es decir, OTP).
- Se deniega varios tipos de *forwarding*. De nuevo, esta opción no aporta seguridad, pero tampoco tiene sentido dejar activadas funcionalidades que no se van a utilizar.

Para que se nos deje de pedir la contraseña a la hora de autenticarnos con SSH hemos de eliminar la línea que realiza esta acción en su PAM (`/etc/pam.d/sshd`). Este cambio se puede ver en el código A.10 (en este caso se ha comentado).

```
[...]
#auth       substack      password-auth
auth       required      pam_google_authenticator.so nullok
[...]
```

Código A.10: Configuración del PAM sshd para la desactivar al autenticación por contraseñas.

También, como se ha cambiado el puerto donde SSH escucha, se ha de cambiar las reglas del firewall. Como se están utilizando los perfiles de aplicaciones de UFW, primero se ha de modificar el puerto en el fichero `/etc/ufw/applications.d/ufw-loginserver` y luego actualizar las reglas mediante el comando:

```
1 $ sudo ufw app update SSH
```

Con todo configurado ya se podrá conectar al servidor remoto utilizando autenticación de clave pública más un código OTP (figura A.4).

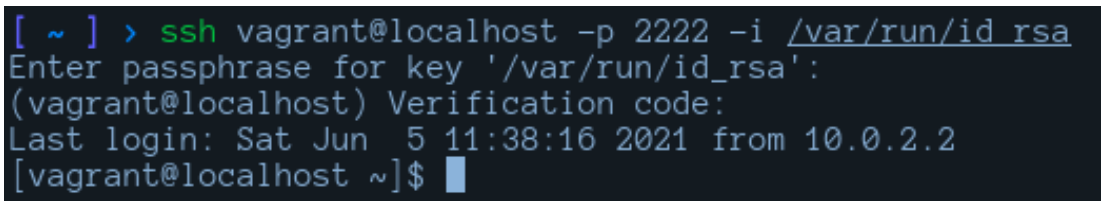


Figura A.4: Autenticación en 2 pasos (clave pública, código OTP) con el servidor SSH.

```
[...]
## Change the port that sshd listens on.
Port 52634
[...]
## Disconnect after time has passed if the user has not
    successfully logged in.
LoginGraceTime 60s

## If enabled, allow root login using ssh.
PermitRootLogin no
[...]
## If enabled, allow password authentication.
PasswordAuthentication no
[...]
## If enabled, allow challenge-response authentication.
ChallengeResponseAuthentication yes

## If enabled enable PAM authentication using
    ChallengeResponseAuthentication
## and PasswordAuthentication.
UsePAM yes

## Specify the authentication methods that must be successfully
    completed for a
## user to grant access.
AuthenticationMethods publickey,keyboard-interactive:pam
[...]
## If enabled, allow ssh-agent fowarding.
AllowAgentForwarding no

## If enabled, allow TCP forwarding.
AllowTcpForwarding no
[...]
```

Código A.11: Fragmento del fichero de configuración del demonio SSH.

A.2.4 Notas finales

A pesar de que en este anexo se han planteado y guiado en la implementación de varias medidas que aumentan la seguridad del servidor de forma sustanciable, todavía se podría mejorar aún más. A continuación se listan algunas de esas posibles mejoras:

- Añadir reglas al tráfico de salida del servidor en lugar de permitirlo todo, teniendo un mayor control sobre el tráfico generado por los servicios corriendo en el sistema.

- Si se supiera con anterioridad desde que localizaciones los usuarios se van a conectar, se podría restringir el acceso al servidor únicamente a esos puntos o localizaciones. En Internet se pueden encontrar múltiples páginas web que mantienen a qué país pertenece qué rango de direcciones IP, pudiendo así prohibir el acceso desde países o continentes enteros.
- Comprobar que las contraseñas de los usuarios del sistema no sean contraseñas comunes ni contraseñas que se puedan encontrar en filtraciones online. Para esto último se podrían utilizar las listas de contraseñas ofrecidas, de forma gratuita, por *haveibeenpwned.com*.
- Aumentar el número de factores de autenticación. Con un nuevo factor, la autenticación podría requerir una clave pública, contraseña y OTP, o dos claves públicas (distintas) y un OTP, por ejemplo.

Lista de acrónimos

API Application Programming Interface. 2, 7, 10, 18, 26, 31, 42, 59, 76

Bash Bourne Again SHell. 49

CD Continuous Delivery. 61

CE Community Edition. 16

CI Continuous Integration. 61

CMS Content Management System. 14

CSS Cascade Style Sheets. 13, 15

CSV Comma-sepparated values. 13, 14

DDoS Distributed Denial-of-Service. 60

DNS Domain Name System. 18, 48, 72

DOI Digital Object Identifier. 37

EE Enterprise Edition. 16

EPEL Extra Packages for Enterprise Linux. 72, 74, 77

GNU GNU's not Unix. 14

GPL General Public License. 14

HOTP HMAC-based One-Time Password. 78

HTML Hypertext Markup Language. 13, 15

HTTP Hypertext Transfer Protocol. 7, 8, 42, 73

HTTPS Hypertext Transfer Protocol Secure. 8, 73

IaC Infrastructure as Code. 9

IP Internet Protocol. 7, 73, 74, 82

JS JavaScript. 15

JSON JavaScript Object Notation. 13, 42

MPM Multi-Proccessing Modules. 7

NIST National Institute of Standards and Technology. 76

OTP One-Time Password. 77–80, 82

PAM Pluggable Authentication Module. 61, 76–79

PDF Portable Document Format. 37

POSIX Portable Operating System Interface. 42

QR Quick Response. 77, 78

REST Representational state transfer. 18

RHEL Red Hat Enterprise Linux. 60

RSS Really Simple Sindication. 14

SSH Secure Shell. 10, 72–74, 77–80

TLS Transport Layer Security. 7

TOTP Time-based One-Time Password. 78

TSV Tab-sepparated values. 13

UFW Uncomplicated Firewall. 72–74, 80

URI Uniform Resource Identifier. 40, 41

URL Uniform Resource Locator. 42

YAML YAML Ain't Markup Language. 10, 13, 16, 18

Glosario

backend Parte de una aplicación que típicamente se ocupa de la lógica de negocio y de la interactuación con bases de datos. Es la contraparte del *frontend*. 9, 40

cluster Grupo de dos o más computadores o nodos que se ejecutan en paralelo para alcanzar un objetivo común. 17

demonio Servicio ejecutándose en segundo plano. 9, 10, 74, 77–79

framework Colección de librerías agrupadas para lograr una solución a un problema común. Por ejemplo, Django es un *framework* web que facilita el desarrollo de aplicaciones web. 6, 7

frontend Parte de una aplicación que típicamente se ocupa de la interacción con el usuario. Es la contraparte del *backend*. 72

widgets Elementos interactivos de una interfaz gráfica de usuario, tales como botones o *scrollbars*. 15

wrapper Aplicación o servicio que actúa como intermediario entre el usuario y la aplicación que oculta. 9, 72

Bibliografía

[1] "The world's fastest framework for building websites | Hugo," accedido: 21-06-2021. [En línea]. Disponible en: https://gohugo.io/

[2] "Why Gatsby?" accedido: 21-06-2021. [En línea]. Disponible en: https://www.gatsbyjs.com/why-gatsby/

[3] "About the Apache HTTP Server Project - The Apache HTTP Server Project," accedido: 21-06-2021. [En línea]. Disponible en: https://httpd.apache.org/ABOUT_APACHE.html

[4] "Module Index - Apache HTTP Server Version 2.4," accedido: 21-06-2021. [En línea]. Disponible en: https://httpd.apache.org/docs/2.4/mod/

[5] "prefork - Apache HTTP Server Version 2.4," accedido: 21-06-2021. [En línea]. Disponible en: https://httpd.apache.org/docs/2.4/mod/prefork.html

[6] "worker - Apache HTTP Server Version 2.4," accedido: 21-06-2021. [En línea]. Disponible en: https://httpd.apache.org/docs/2.4/mod/worker.html

[7] "event - Apache HTTP Server Version 2.4," accedido: 21-06-2021. [En línea]. Disponible en: https://httpd.apache.org/docs/2.4/mod/event.html

[8] "Automatic HTTPS — Caddy Documentation," accedido: 21-06-2021. [En línea]. Disponible en: https://caddyserver.com/docs/automatic-https

[9] "Docker security, Docker daemon attack surface," accedido: 21-06-2021. [En línea]. Disponible en: https://docs.docker.com/engine/security/#docker-daemon-attack-surface

[10] "What is Podman?" accedido: 21-06-2021. [En línea]. Disponible en: https://docs.podman.io/en/latest/

[11] "The Origins of Ansible," accedido: 21-06-2021. [En línea]. Disponible en: https://www.ansible.com/blog/2013/12/08/the-origins-of-ansible

[12] "Red Hat acquires Ansible, the open source IT automation company." accedido: 21-06-2021. [En línea]. Disponible en: https://www.ansible.com/blog/red-hat

[13] "Ansible In Depth, Ansible Whitepaper," accedido: 21-06-2021. [En línea]. Disponible en: https://www.ansible.com/hubfs/pdfs/Ansible-InDepth-WhitePaper.pdf

[14] "Benefits of Agentless, Ansible Whitepaper," accedido: 21-06-2021. [En línea]. Disponible en: https://www.ansible.com/hubfs/pdfs/Benefits-of-Agentless-WhitePaper.pdf

[15] "HAProxy - They use it!" accedido: 21-06-2021. [En línea]. Disponible en: https://www.haproxy.org/they-use-it.html

[16] "HAProxy Forwards Over 2 Million HTTP Requests per Second on a Single Arm-based AWS Graviton2 Instance," accedido: 21-06-2021. [En línea]. Disponible en: https://www.haproxy.com/blog/haproxy-forwards-over-2-million-http-requests-per-second-on-a-single-aws-arm-instance/

[17] "Jekyll | Transform your plain text into static websites and blogs," accedido: 30-05-2021. [En línea]. Disponible en: https://jekyllrb.com/

[18] "Philosophy | Jekyll," accedido: 30-05-2021. [En línea]. Disponible en: https://jekyllrb.com/philosophy/

[19] "Liquid | Jekyll," accedido: 30-05-2021. [En línea]. Disponible en: https://jekyllrb.com/docs/liquid/

[20] "Themes | Jekyll," accedido: 30-05-2021. [En línea]. Disponible en: https://jekyllrb.com/docs/themes/

[21] "Import your old & busted site or blog for use with Jekyll," accedido: 30-05-2021. [En línea]. Disponible en: https://import.jekyllrb.com/

[22] "Showcase | Jekyll," accedido: 30-05-2021. [En línea]. Disponible en: https://jekyllrb.com/showcase/

[23] K. Hornik, "R FAQ," 2020, accedido: 28-05-2021. [En línea]. Disponible en: https://CRAN.R-project.org/doc/FAQ/R-FAQ.html

[24] J. Lai, C. J. Lortie, R. A. Muenchen, J. Yang, and K. Ma, "Evaluating the popularity of R in ecology," 2019, accedido: 28-05-2021. [En línea]. Disponible en: https://esajournals.onlinelibrary.wiley.com/doi/full/10.1002/ecs2.2567

[25] "TIOBE Index for May 2021," accedido: 28-05-2021. Archivado en *The Internet Archive*. [En línea]. Disponible en: https://web.archive.org/web/20210521115101/https://www.tiobe.com/tiobe-index/

[26] "PYPL PopularitY of Programming Language, Worldwide, May 2021," accedido: 28-05-2021. Archivado en *The Internet Archive*. [En línea]. Disponible en: https://web.archive.org/web/20210521142141/https://pypl.github.io/PYPL.html

[27] "The RedMonk Programming Language Rankings: January 2021," accedido: 28-05-2021. [En línea]. Disponible en: https://redmonk.com/sogrady/2021/03/01/language-rankings-1-21/

[28] "Shiny Server Professional v1.5.16 Administrator'S Guide," accedido: 29-05-2021. [En línea]. Disponible en: https://docs.rstudio.com/shiny-server/

[29] "Discontinuation of Sales of Shiny Server Pro to new customers," accedido: 29-05-2021. [En línea]. Disponible en: https://docs.rstudio.com/other/ssp/

[30] "Understanding microservices and Docker – IBM Developer," accedido: 31-05-2021. [En línea]. Disponible en: https://developer.ibm.com/depmodels/microservices/articles/breaking-down-docker-and-microservices/

[31] "What is a Container? | App Containerization | Docker," accedido: 31-05-2021. [En línea]. Disponible en: https://www.docker.com/resources/what-container

[32] "What is Docker? | Microsoft Docs," accedido: 31-05-2021. [En línea]. Disponible en: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/container-docker-introduction/docker-defined

[33] "What is Docker? | IBM," accedido: 31-05-2021. [En línea]. Disponible en: https://www.ibm.com/cloud/learn/docker

[34] "Empowering App Development for Developers | Docker," accedido: 31-05-2021. [En línea]. Disponible en: https://www.docker.com/

[35] "Stack Overflow Developer Survey 2020 - Technology Platforms," accedido: 31-05-2021. [En línea]. Disponible en: https://insights.stackoverflow.com/survey/2020#technology-platforms

[36] "Announcing Docker Enterprise Edition," accedido: 31-05-2021. [En línea]. Disponible en: https://www.docker.com/blog/docker-enterprise-edition/

[37] "Announcing Docker Machine, Swarm, and Compose for Orchestrating Distributed Apps," accedido: 31-05-2021. [En línea]. Disponible en: https://www.docker.com/blog/announcing-docker-machine-swarm-and-compose-for-orchestrating-distributed-apps/

[38] "Docker Hub," accedido: 31-05-2021. [En línea]. Disponible en: https://hub.docker.com/

[39] "Docker Hub - Container Image Library," accedido: 31-05-2021. [En línea]. Disponible en: https://www.docker.com/products/docker-hub

[40] "Orchestration | Docker Documentation," accedido: 31-05-2021. [En línea]. Disponible en: https://docs.docker.com/get-started/orchestration/

[41] "Swarm mode overview | Docker Documentation," accedido: 31-05-2021. [En línea]. Disponible en: https://docs.docker.com/engine/swarm/

[42] "What is NGINX?" accedido: 01-06-2021. [En línea]. Disponible en: https://www.nginx.com/resources/glossary/nginx/

[43] "NGINX vs. Apache: Our View of a Decade-Old Question," accedido: 01-06-2021. [En línea]. Disponible en: https://www.nginx.com/blog/nginx-vs-apache-our-view/

[44] "Usage of web servers broken down by ranking," accedido: 01-06-2021. [En línea]. Disponible en: https://w3techs.com/technologies/cross/web_server/ranking

[45] "NGINX Plus software load balancer, web server, and cache," accedido: 01-06-2021. [En línea]. Disponible en: https://www.nginx.com/products/nginx

[46] "The Scrum Guide," 2020, accedido: 16-06-2021. [En línea]. Disponible en: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf

[47] "Salario de un Programador junior en España," accedido: 19-06-2021. [En línea]. Disponible en: https://es.indeed.com/career/programador-junior/salaries

[48] "Salario de un Jefe de proyecto en España," accedido: 19-06-2021. [En línea]. Disponible en: https://es.indeed.com/career/jefe-de-proyecto/salaries

[49] "Servidor en rack Dell EMC PowerEdge R6515," accedido: 18-06-2021. [En línea]. Disponible en: https://www.dell.com/es-es/work/shop/povw/poweredge-r6515

[50] "Swarm key concepts - Load balancing," accedido: 21-06-2021. [En línea]. Disponible en: https://docs.docker.com/engine/swarm/key-concepts/#load-balancing

[51] "Use multi-stage builds," accedido: 22-06-2021. [En línea]. Disponible en: https://docs.docker.com/develop/develop-images/multistage-build/#use-multi-stage-builds

[52] "CentOS Project shifts focus to CentOS Stream," accedido: 21-06-2021. [En línea]. Disponible en: https://blog.centos.org/2020/12/future-is-centos-stream/

[53] "Isolate containers with a user namespace," accedido: 21-06-2021. [En línea]. Disponible en: https://docs.docker.com/engine/security/userns-remap/

[54] "Mitigating DDoS Attacks with NGINX and NGINX Plus," accedido: 21-06-2021. [En línea]. Disponible en: https://www.nginx.com/blog/mitigating-ddos-attacks-with-nginx-and-nginx-plus/

[55] N. I. of Standards and Technology, "Digital Identity Guidelines - Authentication and Lifecycle Management," U.S. Department of Commerce, Washington, D.C., Tech. Rep. NIST Special Publication 800-63b, 2017, accedido: 05-06-2021. [En línea]. Disponible en: https://doi.org/10.6028/NIST.SP.800-63b