UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Grado en Ingenieria Informatica

Generación automática de ficheros de configuración para el despliegue de servicios en Docker

Automatic generation of configuration files for deploying services in Docker.

Realizado por
Marcos Domínguez Moreno

Tutorizado por
José Miguel Horcas Aguilera

Departamento
Lenguajes y Ciencias de la Comunicación

MÁLAGA, Octubre de 2024

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADA/O EN TECNOLOGÍAS DE LA INFORMACIÓN

Generación automática de ficheros de configuración para el despliegue de servicios en Docker

Automatic generation of configuration files for deploying services in Docker

Realizado por
Marcos Domínguez Moreno

Tutorizado por
José Miguel Horcas Aguilera

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, OCTUBRE DE 2024

Fecha defensa: Diciembre de 2024

Abstract

This work aims to provide a new method for generating configuration files for building software containers. The goal of the project is to create a tool that allows developers to define a configuration and obtain a customized product, such as a *Dockerfile*. By using templates, we can apply best practices for building these files without requiring prior knowledge from the developer. To achieve this, we start by creating templates using *Jinja* syntax and the *UVEngine* variability resolution engine for creating the final product.

The provided tool is a web application that enables developers to select a template and its version to resolve variability and generate a product. Currently, developers can define the configuration using the *UVLS* extension in *VSCode* and select the features of the service they wish to deploy.

The project maintains a repository to store the templates, ensuring they can be updated or accommodate new ones. The system is divided into components to facilitate its maintenance. In conclusion, the effort made allows for the generation of configuration file templates with a high degree of customization compared to other existing methods.

Keywords: Software Product Line, Docker, Configuration Files, Variability Resolution, UVLS, Web Development.

Resumen

Este trabajo busca aportar un nuevo método para la generación de ficheros de configuración para la construcción de contenedores software. El fin del proyecto es generar una herramienta que permita a los desarrolladores definir una configuración y obtener un producto personalizado como puede ser un fichero *Dockerfile*. Utilizando plantillas podemos aplicar sin conocimiento previo del desarrollador las prácticas recomendadas para la construcción de estos ficheros. Para lograr este objetivo se parte de la construcción de plantillas empleando la sintaxis de *Jinja* y el motor de resolución de variablidad *UVEngine* para generar el producto. La herramienta proporcionada es una aplicación web que permite a los desarrolladores seleccionar una plantilla y versión de la misma para resolver la variabilidad y obtener un producto. Actualmente, los desarrolladores pueden definir la configuración haciendo uso de la extensión *UVLS* en *VSCode* y seleccionar las características del servicio que desean desplegar. El proyecto mantiene un repositorio para almacenar las plantillas con el objetivo de que estas puedan ser actualizadas o bien dar cabida a nuevas plantillas. El sistema se ha dividido en componentes para facilitar el mantenimiento del mismo. En , el esfuerzo realizado permite generar plantillas de ficheros de configuración con un alto grado de personalización en comparación con otros métodos existentes.

Palabras clave: SPL, Docker, Despliegues Servicios, Desarrollo web, Configuración, feature models, Línea de Productos

Índice

1 Introducción

1.1. Motivación

Las herramientas de automatización de tareas son fundamentales para la productividad en el desarrollo de software, permiten mejorar la eficiencia, la calidad y la consistencia en la producción de código. Al relegar tareas repetitivas y propensas a errores a herramientas automatizadas, los desarrolladores pueden centrarse en tareas más creativas y de mayor valor añadido.

Parte del ciclo de vida en un desarrollo de software implica la configuración y el despliegue de servicios en entornos de producción y desarrollo. La configuración y despliegue requiere de la creación de archivos de configuración específicos, como son, por ejemplo, los archivos *Dockerfile* y *.dockerignore*, que definen cómo se construye la imagen de un contenedor en tecnologías como *Docker*.

En el contexto de la tecnología de contenedores, Docker se ha convertido en una de las plataformas más populares para la creación, el despliegue y la gestión de aplicaciones en entornos contenerizados. Sin embargo, la configuración de servicios en Docker puede ser una tarea compleja y propensa a errores, especialmente cuando se manejan múltiples servicios y configuraciones.

La motivación para realizar este proyecto surge de la necesidad de simplificar y automatizar el proceso de generación de archivos de configuración para el despliegue de servicios en Docker. En la actualidad, la creación manual de estos archivos Dockerfile puede ser una tarea repetitiva y susceptible a fallos al no conocer el usuario de toda la información necesaria para la correcta configuración de los servicios.

Al desarrollar una herramienta que automatice este proceso, se busca mejorar la eficiencia y la precisión en la configuración de entornos de contenedores, facilitando así el trabajo de

desarrollo y administración.

1.2. Objetivo

La meta principal de este proyecto es desarrollar una herramienta que permita la generación automática de archivos de configuración para el despliegue de servicios en Docker. Siguiendo las pautas recomendadas para la generación de dichos archivos mediante de plantillas, las plantillas han de ser extensibles y adaptables a los cambios en un futuro sin necesidad de reescribir código.

Para ello se plantean los siguientes objetivos específicos:

- Investigar y analizar las tecnologías y herramientas existentes para la generación de archivos de configuración en Docker.
- Desarrollar una plantilla orientada a la generación de ficheros Dockerfile siguiendo las buenas prácticas.
- Implementar un modelo de plantillas extensible y adaptable a los cambios en las tecnologías y las metodologías de desarrollo.
- Desarrollar una herramienta web que permita la selección de plantillas y la generación de archivos de configuración para el despliegue de servicios en Docker.
- Desplegar la aplicación en un entorno no orientado a producción y documentar el proceso de instalación y uso.
- Evaluar la eficacia y la usabilidad de la herramienta en la memoria.

1.3. Resultados Esperados

Dada la gran cantidad de opciones de configuración disponibles para la generación de ficheros de configuración, el desarrollo de esta herramienta presenta varios desafíos técnicos y conceptuales. El primero de ellos es la magnitud de las opciones de configuración disponibles, que pueden variar en función de las necesidades y los requisitos de cada servicio. dar una cobertura a todos los escenarios posibles es una tarea inabarcable, por lo que se ha optado por centrarse en los casos de uso más comunes y en las configuraciones más utilizadas en la práctica. Sin embargo, se espera que la herramienta sea lo suficientemente flexible y extensible como para permitir la incorporación de nuevas plantillas y configuraciones en el futuro.

Los desarrollos de los servicios resultantes de la implementación no tendrán el objetivo de alcanzar un estado de producción sino de servir como base para futuras implementaciones y desarrollos orientados a ello.

1.4. Estructura del documento

- El Capítulo 1 describe la motivación del trabajo, sus objetivos, y los resultados esperados.
- El Capítulo 2 presenta una breve introducción a los contenidos, al estado del arte, soluciones existentes y lo que se aporta en este contexto.
- El Capítulo 3 describe las tecnologías y herramientas empleadas en el desarrollo del proyecto.
- El Capítulo 4 describe la metodología de trabajo empleada en el desarrollo del proyecto.
- El Capítulo 5 desarrolla el modelado de las plantillas.
- El Capitulo 6 define la arquitectura del proyecto y junto con la descripción de los servicios
- El Capítulo 7 es un capítulo dedicado a la extensibilidad de las plantillas
- El Capítulo 8 describe el despliegue de la aplicación en un entorno no orientado a producción.
- El Capítulo 9 presenta los resultados obtenidos en el desarrollo del proyecto.

- El Capítulo 10 incluye las conclusiones del trabajo y las recomendaciones para trabajos futuros.
- El Apéndice A Es un manual de Instalación de la aplicación usando contenedores Docker.
- El Apéndice B Es un manual de Uso de la aplicación web.
- El Apéndice C Es una guía para el desarrollo de plantillas, nuevas y existentes.

2

Estado del Arte

Para comprender el sentido de este proyecto y su problemática es necesario contextualizar algunos conceptos previos que son necesarios para comprender la situación actual y reconocer los posibles problemas. A continuación se introducen dichos conceptos de manera escalonada, abordaremos la problemática de la generación de archivos de configuración en Docker, las tecnologías empleadas en la generación de dichos archivos y las soluciones actualmente disponibles.

2.1. Contenedores

Los contenedores son paquetes software ligeros que incluyen el código de las aplicaciones junto con sus dependencias, como versiones concretas de entornos de ejecución de ciertos lenguajes de programación y bibliotecas para ejecutar los servicios de software [17].

2.1.1. ¿Por qué se usan los contenedores?

Los contenedores constituyen un mecanismo de empaquetado lógico en el que las aplicaciones pueden extraerse del entorno en que realmente se ejecutan. Esta desvinculación facilita el despliegue uniforme de las aplicaciones basadas en ellos con independencia de que el entorno sea un centro de datos privado, la nube pública o el portátil personal de un desarrollador [17]. Es una solución de fácil aplicación, alto rendimiento y extremadamente portátil.[1].

[1]Existen restricciones en cuanto a la portabilidad de los contenedores, pero en general son fácilmente desplegables en cualquier entorno enlace:https://stackoverflow.com/questions/42158596/can-windows-containers-be-hosted-on-linux

2.2. Imágenes

Las imágenes de contenedores son plantillas de solo lectura que contienen el código de la aplicación, las bibliotecas, las dependencias y otros archivos necesarios para ejecutar un contenedor [1]. No necesariamente necesitamos de una imagen para construir un contenedor[2] pero ese es el escenario más común.[3]

2.3. Docker

Docker [11] es una plataforma de código abierto que permite a los desarrolladores crear, desplegar y ejecutar aplicaciones en contenedores. Docker automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos.[4]

2.3.1. Dockerfile

El archivo Dockerfile [**Dockerfile_concepts**] es un archivo de texto que contiene una serie de instrucciones que Docker utilizará para construir una imagen de contenedor. Usualmente, la construcción de un archivo de configuración se realiza a partir de una imagen base, que puede ser una imagen oficial de Docker o una imagen personalizada creada por el usuario. A dicha imagen se le añaden las instrucciones necesarias para instalar las dependencias y configurar el entorno de ejecución de la aplicación, lo que viene a representar el contenido de un archivo Dockerfile.

2.3.2. Dockerignore

El fichero .dockerignore [22] es un tipo de archivo que puede colocarse en el proyecto del usuario y que se utiliza con la función de ignorar ficheros y directorios durante el `docker build`, con el objetivo de evitar que estos se copien a la imagen de Docker.

[2] Existe la posibilidad de crear contenedores de otras formas, como a partir de otros contenedores.

[3] Construir un contenedor a partir de una imagen lo hace reproducible.

[4] Existen alternativas a Docker como Podman [35] pero Docker es la más popular y ampliamente utilizada en la actualidad.

2.3.3. Buenas Prácticas en la creación de las imágenes

Para la creación de los ficheros Dockerfile es recomendable seguir una serie de buenas prácticas para garantizar la eficiencia y la seguridad en el despliegue de servicios en Docker. Estas prácticas se recogen en la documentación oficial de Docker [**Dockerfile_best_practices**] En la documentación se abordan una serie de recomendaciones para la creación de imágenes de contenedores en Docker, entre las que se incluyen:

- Utilizar imágenes oficiales de Docker:
  - Las imágenes oficiales son mantenidas y actualizadas por la comunidad de Docker, lo que garantiza un alto nivel de seguridad y estabilidad.
  - Estas imágenes suelen estar optimizadas para un rendimiento eficiente y son ampliamente documentadas.
- Crea ficheros de configuración Multietapa:
  - Crear imágenes en varias etapas permite dividir la construcción de la imagen, lo que puede mejorar el rendimiento.
  - Esto ayuda a reducir el tamaño final de la imagen, incluyendo solo los archivos necesarios para la ejecución de la aplicación.
- Usar una imagen base ligera:
  - Utilizar imágenes base ligeras, como `'alpine'`, puede reducir significativamente el tamaño de la imagen Docker.
  - Las imágenes ligeras también suelen tener menos vulnerabilidades de seguridad debido a su menor superficie de ataque.
- Reducir el número de capas:
  - Cada instrucción en un Dockerfile crea una nueva capa en la imagen. Reducir el número de capas puede mejorar el rendimiento y reducir el tamaño de la imagen.
  - Combinar múltiples comandos en una sola instrucción 'RUN' puede ayudar a minimizar el número de capas.

- Eliminar archivos innecesarios:
    - Durante el proceso de construcción, es importante eliminar cualquier archivo temporal o innecesario para mantener la imagen lo más pequeña posible.
- Evitar el uso de la etiqueta `latest`:
    - La etiqueta `latest` puede ser ambigua y llevar a problemas de consistencia y reproducibilidad.
    - Es mejor especificar una versión concreta de la imagen para asegurar que se está utilizando la versión correcta.
- Eliminar archivos temporales y de instalación:
    - Durante la instalación de paquetes y dependencias, se generan archivos temporales que deben ser eliminados para reducir el tamaño de la imagen.
    - Utilizar comandos como ‘`apt-get clean`’ y ‘`rm -rf /var/lib/apt/lists/*`’ para limpiar los archivos de instalación.

[5]

[5]Para más información: https://docs.docker.com/build/building/best-practices/

2.4. Problemática

La dificultad de crear imágenes reside en la naturaleza de la aplicación que queremos llevar a un contenedor. Existen una gran cantidad de escenarios donde, si bien la tarea es sencilla, esta puede esconder una complejidad implícita de la que el usuario no es consciente. Estos apartados repercuten en el tiempo de compilación, tamaño de la imagen, mantenibilidad, seguridad y repetibilidad en las imágenes que se crean.
La calidad de las imágenes pueden marcar una diferencia que puede llegar a ser crítica para posteriores etapas en el desarrollo de aplicaciones, como son la Orquestación de contenedores [21].

2.5. Soluciones Actualmente Disponibles

En la actualidad hay disponibles varias soluciones para disminuir la complejidad implícita que reside en la configuración de dichos ficheros Dockerfile. Existen más herramientas que las representadas a continuación, pero estas son las más empleadas.

2.5.1. Docker init

Docker init es el comando de Docker Desktop [8] que permite inicializar un proyecto con un archivo de configuración básico. Su funcionamiento consta de elegir una plantilla de proyecto y seleccionar entre las opciones de configuración. Las opciones de configuración se limitan a elegir el lenguaje de programación, puerto en el que despliega la aplicación y que comando se ejecutara al iniciar el contenedor. Las plantillas son creadas y mantenidas por la comunidad.[6]

[6]En este proyecto se reutilizan y se recogen ideas de varias de las plantillas que incluye Docker Init

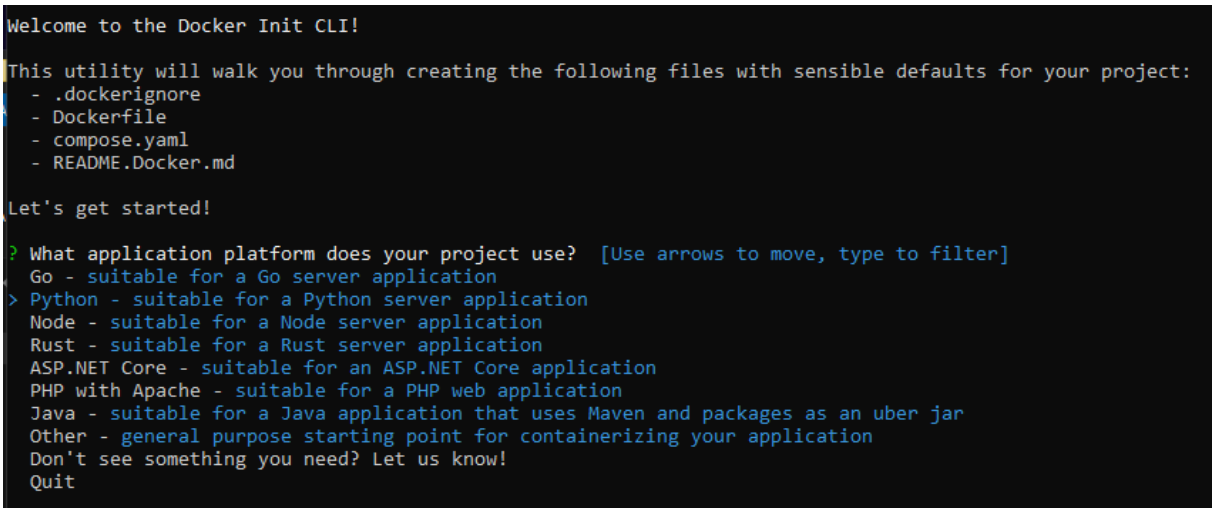


Figura 1: Captura de las opciones de Plantillas disponibles Docker Init.

2.5.2. Docker Extension for Visual Studio Code

La extensión de Docker para Visual Studio Code [5] permite entre otras muchas de sus funcionalidades, la generación del fichero Dockerfile entre otros. Su comportamiento es similar al de Docker init, permitiendo, entre otras opciones, elegir el lenguaje de desarrollo del proyecto y qué comando se ejecutará al iniciar el contenedor.

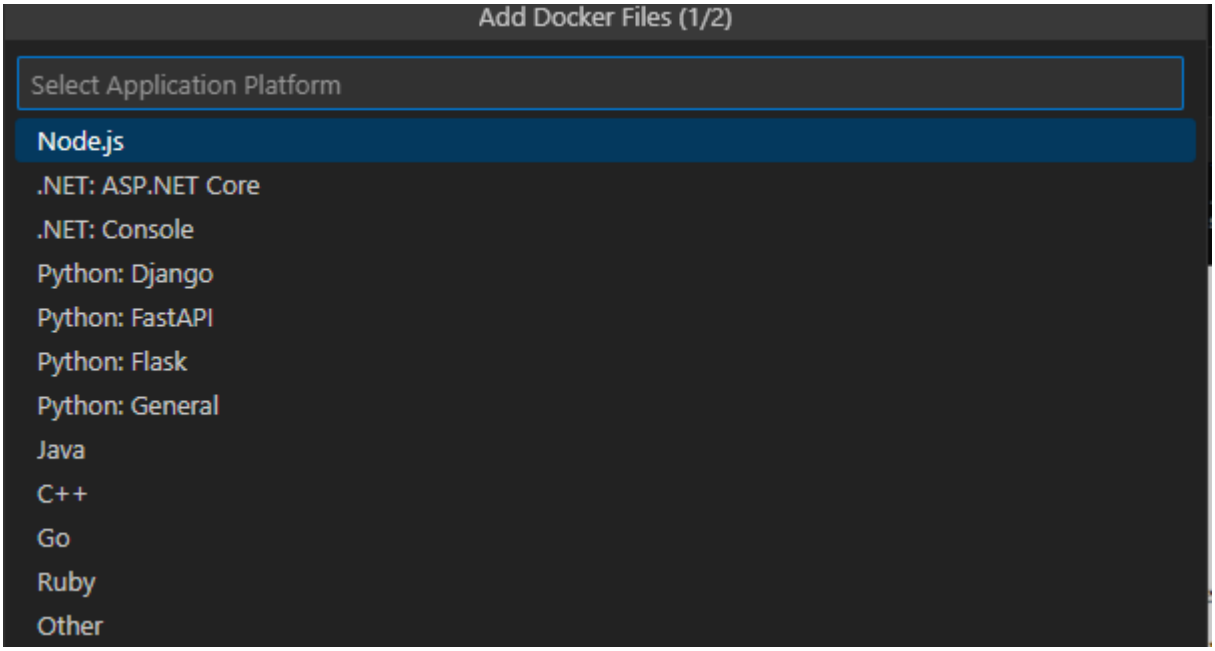
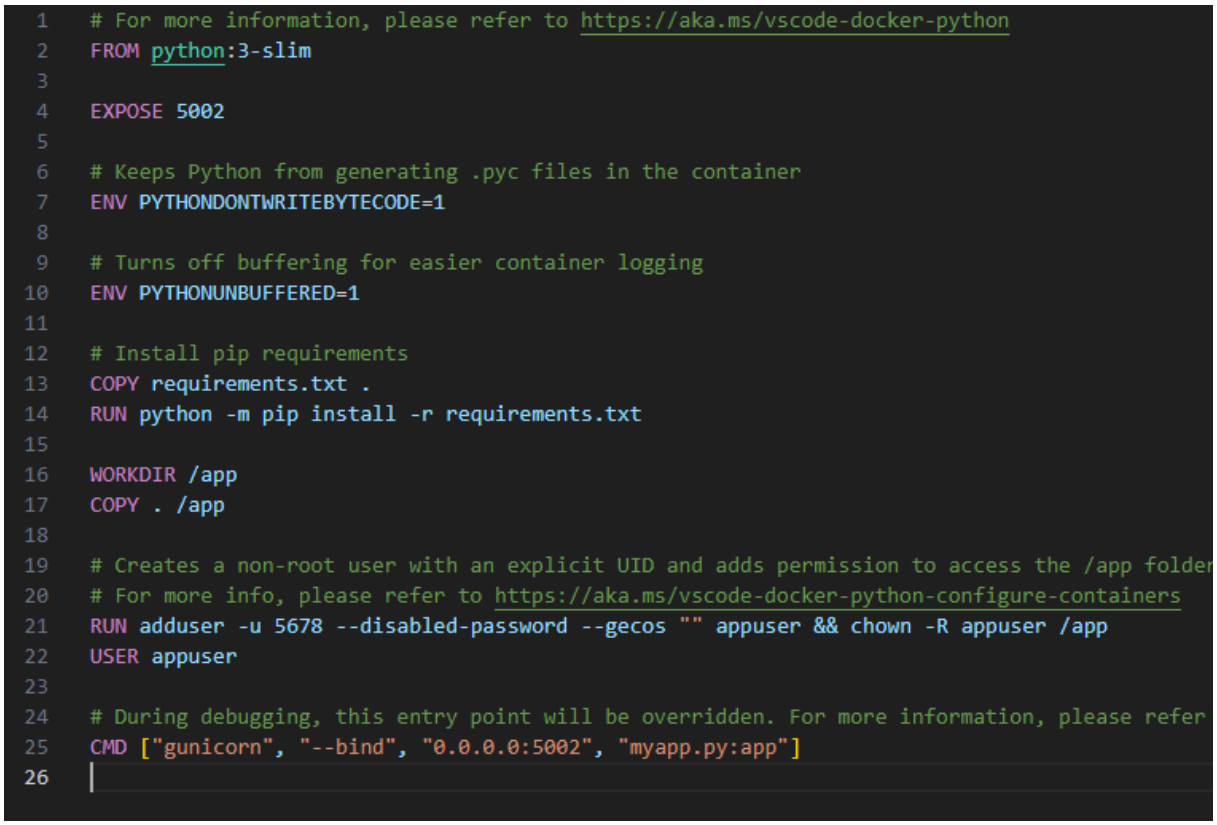


Figura 2: Plantillas Disponibles en la extensión para Docker de Visual Studio Code.



```
# For more information, please refer to https://aka.ms/vscode-docker-python
FROM python:3-slim

EXPOSE 5002

# Keeps Python from generating .pyc files in the container
ENV PYTHONDONTWRITEBYTECODE=1

# Turns off buffering for easier container logging
ENV PYTHONUNBUFFERED=1

# Install pip requirements
COPY requirements.txt .
RUN python -m pip install -r requirements.txt

WORKDIR /app
COPY . /app

# Creates a non-root user with an explicit UID and adds permission to access the /app folder
# For more info, please refer to https://aka.ms/vscode-docker-python-configure-containers
RUN adduser -u 5678 --disabled-password --gecos "" appuser && chown -R appuser /app
USER appuser

# During debugging, this entry point will be overridden. For more information, please refer
CMD ["gunicorn", "--bind", "0.0.0.0:5002", "myapp.py:app"]
```

Figura 3: Captura de una plantilla generada por la extensión para Docker de Visual Studio Code.

2.5.3. Inteligencia Artificial

En la actualidad existen soluciones que se apoyan en la Inteligencia Artificial (IA) para la generación de archivos de configuración y para la creación de los ficheros Dockerfile. La calidad de estas soluciones es variable y depende de la calidad de los modelos de lenguaje natural empleados y de la calidad del *prompt*.[7] de entrada como *magickpen* [**magickpen_Dockerfile_template**][8] o alternativas con propósito más general como *chatgpt* [34] [9]

[7] Instrucción o texto inicial proporcionado a una herramienta generativa de IA para dirigir la generación de respuestas o resultados específicos [23].

[8] magickpen: https://magickpen.com/templates/128/.

[9] chatgpt: https://chat.openai.com/.

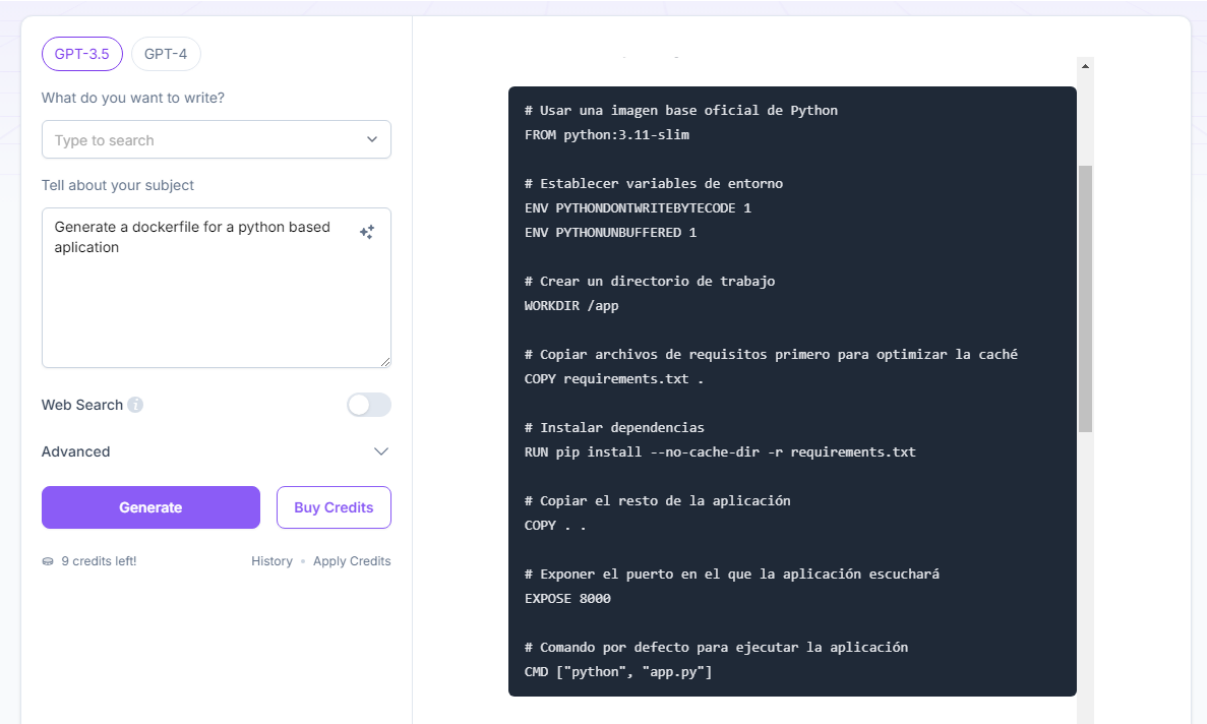


Figura 4: Captura de un resultado de *magickpen*.

2.6. Conclusiones Preliminares

En este estudio se han recogido algunas de las principales herramientas y tecnologías disponibles en la actualidad para la generación de archivos de configuración. Todas las herramientas (salvo propiamente los Modelos de Texto Generativo) hacen uso de plantillas para la generación de los ficheros Dockerfile y dockerignore, estas plantillas, si bien son correctas, son por naturaleza limitadas y no permiten la personalización final de las opciones de configuración. No obstante, suponen un punto de partida para el desarrollo de aplicaciones contenerizadas.

2.7. ¿Qué se aporta en este contexto?

Las opciones actualmente disponibles para la generación de archivos de configuración para el despliegue de servicios en Docker son limitadas y no permiten la personalización de las opciones de configuración. Se modela de forma separada la variabilidad y las posibles opciones que existen a la hora de construir las imágenes. El proyecto proporciona una nueva forma de construir plantillas dotando capacidad al usuario de una personalización más profunda y

detallada de las opciones de configuración de los servicios en Docker.

2.7.1. Líneas de Producto Software (SPL)

En el proyecto tendremos que construir nuestras propias plantillas que formarán parte de una línea de productos software. Una línea de productos software (SPL del inglés *Software Product Line*) es una familia de productos software relacionados entre sí que comparten ciertas características comunes (similitudes) pero que también tienen características variables.

En una SPL, los sistemas se descomponen en características (*features*) que representan funcionalidades o comportamientos del sistema. El principal artefacto para modelar la variabilidad en SPLs son los modelos de variabilidad. Existen muchos modelos de variabilidad, pero los más extendidos y usados en la práctica son los modelos de características (*feature models*).

Un *feature model* (FM) [**Kang1990_FODA**, 43] es un modelo para representar la variabilidad de una SPL en base a características comunes y variables (véase Figura 1), especificando qué características se pueden seleccionar en una configuración para generar un producto válido de la SPL.

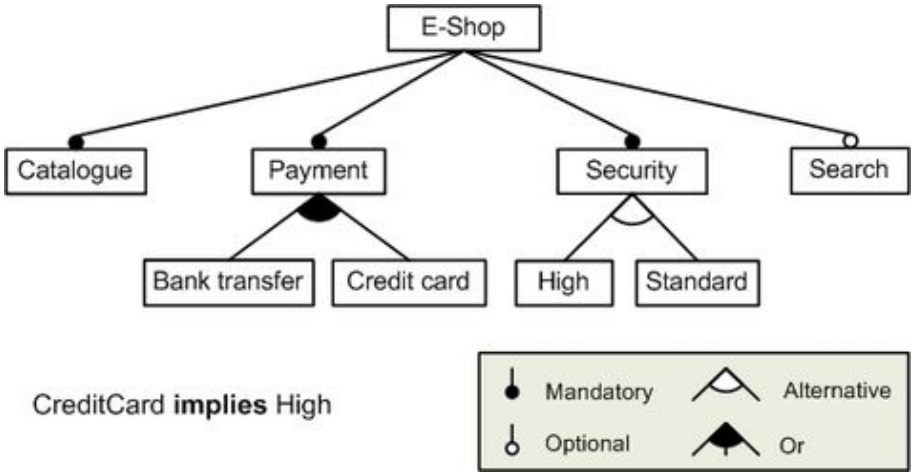


Figura 5: Ejemplo de representación gráfica de un *feature model* [43].

3

Tecnologías Empleadas

Este apartado describe las tecnologías y herramientas empleadas más relevantes para el desarrollo del proyecto, así como su función y su relación con el objetivo del proyecto.

3.1. Universal Variability Language (UVL)

Universal Variability Language (en adelante UVL) [6] es un lenguaje de modelado diseñado específicamente para representar y manejar la variabilidad. La variabilidad en software se refiere a las partes de un sistema que pueden personalizarse, configurarse o cambiarse para adaptarse a diferentes necesidades o contextos. UVL se usa comúnmente en el contexto de ingeniería de líneas de productos software, donde se trabaja con una familia de productos que comparten un núcleo común, pero pueden tener variaciones. Todas las plantillas que se han desarrollado en este proyecto requieren de un archivo UVL para poder generar configuraciones válidas.

3.2. Docker

La aplicación web, tanto como todos sus servicios, han sido desarrollados con el objetivo en mente de ser ejecutados dentro de contenedores. Para el manejo de los contenedores en el entorno de desarrollo se ha optado por la utilización de Docker Desktop, un complemento con la consola de comandos. Esta herramienta permite al proyecto hacerlo portable y fácil de desplegar en cualquier entorno.

3.3. Node.js

[31] para el desarrollo de la aplicación web, se ha optado por la utilización de Node.js como entorno de ejecución de JavaScript para el desarrollo del Frontend. Esta herramienta constituye una plataforma de desarrollo en tiempo de ejecución que tiene la ventaja de contar con una extensa biblioteca de módulos que simplifican el desarrollo de aplicaciones web. Con npm (Node Package Manager) [33], Node.js ofrece acceso a una vasta colección de paquetes y módulos que pueden ser fácilmente integrados en el proyecto, acelerando el desarrollo y reduciendo la necesidad de escribir código desde cero. Para el proyecto se emplea la versión v22.10.0 y la versión v10.9.0 del gestor de paquetes npm

3.4. Angular

[2] En cuanto al desarrollo de la aplicación web, se ha optado por la utilización de Angular como framework de desarrollo. El motivo de esta decisión es la idea de emplear un framework moderno que se usa con los métodos de desarrollo actuales y las mejores prácticas en el desarrollo de aplicaciones web. Emplear otros frameworks de desarrollo como React o Vue.js también hubiera sido una opción válida al existir una cierta convergencia entre ellos en cuanto a funcionalidades y características. Para los requisitos del proyecto entendimos que Angular ya integra gran parte de las funcionalidades que requería nuestra aplicación ahorrando tiempo en la implementación de las mismas. La versión empleada es Angular v18.[10]

3.4.1. Angular Material

[3] Para el diseño de la interfaz gráfica de la aplicación web se ha optado por la utilización de los componentes provenientes de los módulos de Angular Material. Siguiendo la línea de diseño de Material Design, Angular Material proporciona una serie de componentes y directivas que facilitan la creación de interfaces de usuario modernas y atractivas. Los componentes tienen una interfaz conocida por el usuario y son fáciles de implementar en la aplicación web.[11]

[10] Angular 18 no es la última versión disponible, pero al depender de otros desarrollos, se ha optado por la utilización de esta versión.

[11] La última versión disponible es 18.2.9.

3.4.2. Monaco Editor

[28] Para la edición de los archivos de configuración se ha optado por la utilización de Monaco Editor como editor de código. Monaco Editor es un editor de código fuente basado en la web que se utiliza en varios proyectos de Microsoft, como Visual Studio Code, Azure DevOps y el editor de código de GitHub, cuenta con una serie de características que lo hacen ideal para la edición de archivos de configuración, como el resaltado de sintaxis, el autocompletado y la verificación de errores. también es altamente personalizable y extensible, lo que permite adaptarlo a las necesidades del proyecto.[12]

3.4.3. ngx-json-viewer

[32] Para la visualización de los archivos de configuración se ha optado por la utilización de ngx-json-viewer como visor de archivos JSON. ngx-json-viewer es un componente de Angular que permite visualizar archivos JSON en un formato legible y estructurado. Es un proyecto de código abierto que se puede integrar fácilmente en aplicaciones Angular y personalizar según las necesidades del proyecto.

3.5. Nginx

[30] Nginx es un servidor web de código abierto desarrollado en C que se utiliza para servir contenido web estático y dinámico, así como para actuar como proxy inverso y balanceador de carga. la ventaja de Nginx es principalmente la poca necesidad de configuración que requiere para su correcto funcionamiento, además de ser un servidor web ligero y de alto rendimiento. Para las necesidades del proyecto, se ha empleado en sus funciones de proxy inverso para redirigir las peticiones de la aplicación web al servidor de backend y como servidor web para servir el contenido estático de la aplicación web.

3.6. Python

[13] Python es un lenguaje de programación de alto nivel, interpretado y de propósito general que se ha convertido en uno de los lenguajes más populares en la actualidad. Para el

[12]La última versión de Angular con la que es compatible es la v18 por este motivo se ha desarrollado en esta versión de Angular https://www.npmjs.com/package/ngx-monaco-editor-v2.

desarrollo de los servicios de backend se ha optado por la utilización de Python como lenguaje de programación. Python es conocido por su sintaxis clara y legible, su amplia biblioteca estándar y su soporte para múltiples paradigmas de programación, como la programación orientada a objetos, la programación funcional y la programación imperativa.

3.6.1. Flask

[36] Flask es un microframework de Python para el desarrollo de aplicaciones web. Flask es conocido por su simplicidad y su facilidad de uso, lo que lo convierte en una excelente opción para el desarrollo de aplicaciones web pequeñas y medianas.

3.6.2. Gunicorn

[18] Gunicorn es un servidor HTTP WSGI para Python que se utiliza para servir aplicaciones web desarrolladas en Python. Su objetivo es proporcionar un servidor web simple y eficiente que pueda manejar múltiples solicitudes simultáneamente. Esta herramienta es necesaria para el despliegue de la aplicación en un entorno de producción.

3.6.3. Pypi

[38] El uso de Pypi como repositorio de paquetes de Python es necesario para la instalación de las dependencias de la aplicación. Además, ha sido empleado para subir paquetes diseñados para la aplicación.

3.7. Jinja

[37] Jinja es un motor de plantillas para Python que se utiliza para generar archivos de texto dinámicos basados en plantillas.

3.7.1. Jinja2

Jinja2 es la versión más reciente de Jinja y se utiliza comúnmente en aplicaciones web de Python para generar contenido dinámico. Jinja2 es conocido por su sintaxis simple y legible, su capacidad para generar contenido dinámico y su integración con Python y otros marcos de desarrollo web. Jinja2 permite la generación de archivos de texto dinámicos basados en

plantillas, lo que facilita la creación de archivos de configuración personalizados y adaptables a las necesidades del usuario. Todas las plantillas desarrolladas han usado la sintaxis de Jinja2.

3.8. Git

[40] Para el control de versiones se ha optado por la utilización de Git como sistema de control de versiones.

3.9. GitHub

[15] GitHub es una plataforma de desarrollo colaborativo que se utiliza para alojar proyectos de código, realizar seguimiento de problemas y tareas, y colaborar con otros desarrolladores. Todo el proyecto es alojado en GitHub incluyendo plantillas, código, y demás recursos del proyecto.

3.9.1. GitHub Public Api

[16] Para la gestión de las plantillas se ha empleado la API pública de GitHub para la obtención de las plantillas disponibles. Esta decisión se ha tomado por facilidad de uso, ya que la API de GitHub es ampliamente conocida y documentada, y permite acceder a una gran cantidad de información sobre los repositorios y las plantillas disponibles. Esto supone que las plantillas están visibles y son accesibles y modificables por cualquier usuario de GitHub. Simplificando el proceso de creación, mantenimiento y actualización de las plantillas.

3.10. Visual Studio Code

[29] Visual Studio Code es un editor de código fuente desarrollado por Microsoft que se utiliza para escribir, editar y depurar código. Su mención se debe a que es el editor de código fuente que se ha empleado para el desarrollo del proyecto. Además de ser clave en el proceso como generador de ficheros de configuración de los que la aplicación hace uso.

3.10.1. UVLS extension

UVLS[4] es una extensión de Visual Studio Code que proporciona soporte para UVL en el editor de código. La extensión UVLS proporciona resaltado de sintaxis, autocompletado, verificación de errores y otras funcionalidades que facilitan la edición de archivos de configuración en UVL. Además, la clave más importante es que es la única herramienta existente que permite crear una configuración correcta partiendo desde un *feature model*.

Otras extensiones de menos importancia empleadas en *VSCode* son:

- **Docker Extension**
- **Flama**
- **Graphviz Interactive Preview**
- **Latex Workshop**
- **Pylance**
- **Python Debugger**
- **Better Jinja**

4

Metodología del trabajo

4.1. Introducción

El enfoque que se ha dado para abordar el desarrollo de este trabajo ha consistido en combinar un trabajo previo de análisis e investigación con una fase de desarrollo iterativo.

La investigación y análisis tenía el objetivo de conocer en profundidad las tecnologías y herramientas empleadas para posteriormente estudiar una posible aplicación de estas ideas y conceptos en el desarrollo de las plantillas. Una vez resuelta esta primera parte, se ha procedido a una toma de requisitos y a la planificación de la implementación de la aplicación. Para la implementación de la aplicación, se ha optado por la utilización de una metodología de desarrollo que permita la evolución del proyecto a un producto mínimo viable en el menor tiempo posible.

4.2. Metodología del Desarrollo

Para la implementación del Proyecto se ha optado por un desarrollo en Cascada al tener una idea muy definida de lo que se quiere desarrollar y de las tecnologías empleadas. Los cambios más relevantes surgen en las plantillas, es por ello que contamos con los mecanismos necesarios para adaptarnos a estos cambios sin que repercutan en el resto de la aplicación. La implementación en cascada propone una serie de fases que se deben seguir de forma secuencial, pero para este escenario cada fase se inicia sin la necesidad de que la anterior haya sido completada, de forma que permita adaptarnos a requerimientos no planeados y encajar nuevas ideas que pudieran sucederse. En todo momento el código es susceptible de cambios.

4.2.1. Justificación

Al encontrarnos ante un trabajo individual, toda la responsabilidad y capacidad del desarrollo recae sobre una persona; esto supone una limitación en cuanto al contenido, pero supone una ventaja a la hora de adaptar cambios de la aplicación al centralizar todo el desarrollo en una persona.

El tiempo disponible es limitado por naturaleza y es vital simplificar todas las fases de la planificación del desarrollo. Por este motivo tenemos que descartar metodologías de desarrollo iterativo como Scrum y centrarnos en metodologías más rígidas como un desarrollo en Cascada.

El desarrollo de las plantillas requiere un estudio con el fin de limitar las opciones de configuración y centrarse en los casos de uso más comunes y en las configuraciones más utilizadas en la práctica.

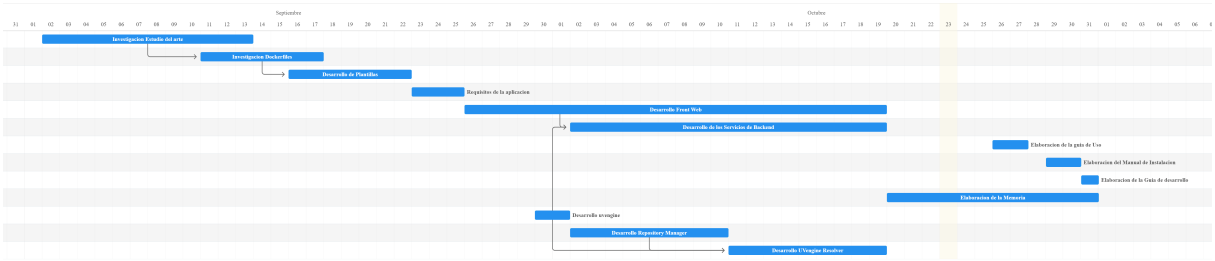


Figura 6: Diagrama de Gantt (generado por la herramienta *Diagrama de Gantt Online*).

4.2.2. Notas del Procedimiento

Se evaluaron con el tutor los momentos más relevantes, como los requisitos de la aplicación, arquitectura y una evaluación final, para introducir cambios que resultaron necesarios en tanto en el proyecto como en la documentación y en las plantillas.

4.2.3. Limitaciones

El desarrollo de un proyecto de estas características requiere de una planificación y una metodología de trabajo adecuadas. En este sentido, como se ha mencionado anteriormente, se identificaron una serie de limitaciones y restricciones que pueden afectar al desarrollo del

proyecto. En este sentido, el desarrollo de las plantillas tuvo que limitarse en cuanto a formato y contenido, ya que no se pueden abarcar todas las posibles opciones de configuración.
El tiempo que se puede dedicar al proyecto tiene que estar dentro de unos límites razonables para un trabajo de estas características y no se puede extender indefinidamente.

4.3. Fases de Estudio

4.4. Estudio del Estado del Arte:

En esta fase se realiza una revisión exhaustiva de la literatura y tecnologías existentes relacionadas con el tema del proyecto. El objetivo es comprender el panorama actual, identificar tendencias, mejores prácticas y tecnologías relevantes que puedan influir en el desarrollo del proyecto. Los puntos clave de esta fase son:

- Para comprender la tecnología de los contenedores fue esencial la lectura de Modern Infrastructures and Applications with Docker. [27]
- Completar la introducción al lenguaje UVL con UVL Playground [42].

4.5. Investigación diseño de Dockerfiles

Esta fase implica investigar y evaluar las diferentes opciones de configuración que se presentan en el desarrollo de un fichero Dockerfile en el despliegue de aplicaciones. Se exploran las características, herramientas y técnicas que pueden optimizar el uso de Docker en un proyecto software.

- Para comprender la importancia de las buenas prácticas en la creación de imágenes, fue necesario la lectura de la documentación oficial de Docker Dockerfile reference [**Dockerfile_reference**].

4.6. Desarrollo de Plantillas

Esta fase implica la construcción del árbol de características y las restricciones textuales que orienten el espacio de configuraciones posibles del fichero Dockerfile. En este paso se ponen en práctica todo el conocimiento adquirido en los pasos anteriores que y como

resultado se obtiene el modelado de unas plantillas. Hemos de tener en cuenta las buenas prácticas y las recomendaciones de la comunidad para la creación de los ficheros Dockerfile [**Dockerfile_concepts**]. Estas prácticas tienen que estar presentes en el diseño de las plantillas desde el inicio, también hemos de generar test para mantener la consistencia de las implementaciones. Esta fase queda cubierta con más detalle en el Capítulo 5.

4.7. Fase Implementación

4.7.1. Requisitos de la aplicación

En esta sección se documentan los requisitos funcionales de la aplicación, obviando los requisitos no funcionales. Los requisitos han cambiado conforme el proyecto avanzaba en sus etapas de desarrollo sin ánimo de ser exhaustivos las funcionalidades mínimas que se pretenden alcanzar en cuanto a la generación de productos son:

- La herramienta debe permitir seleccionar la plantilla y la versión con la que se desea trabajar.
- Dentro de la aplicación se podrá recargar con nuevas plantillas que configuremos en un repositorio.
- La herramienta debe generar el producto en el momento en el que se aplique una configuración válida.
- La herramienta debe permitir la descarga o copia al portapapeles del producto generado.
- La herramienta debe permitir la visualización de los archivos de configuración generados.
- La herramienta debe permitir la edición de los archivos de configuración generados.
- La herramienta debe de ser una solución portable a otros sistemas operativos.

4.7.2. Diseño de la aplicación

En esta fase se diseña la arquitectura de la aplicación, se definen los componentes y módulos que la componen y se establecen las relaciones entre ellos. En esta imagen se resumen los componentes de la aplicación. Una explicación más detallada puede encontrarse en el Capítulo 6

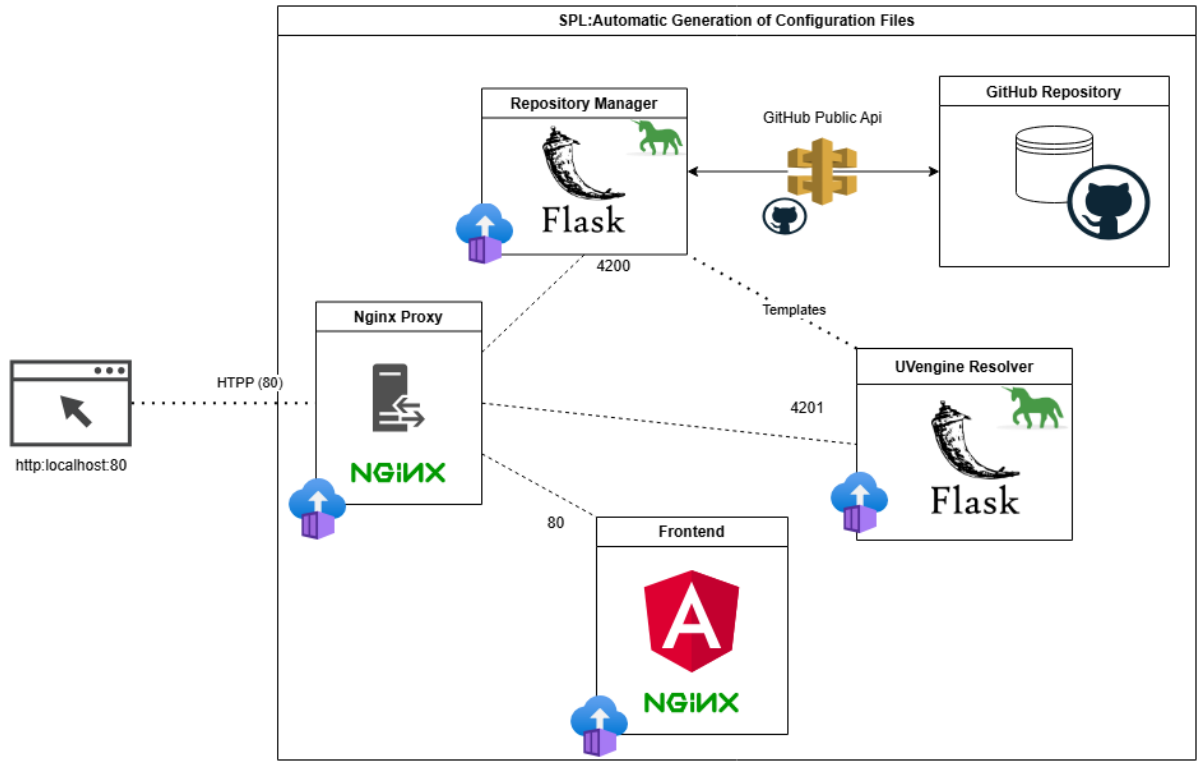


Figura 7: Diagrama de infraestructura.

4.7.3. Desarrollo del Paquete UVEngine

En esta fase se adapta el proyecto de la aplicación de UVEngine [19], que permite la generación automática de archivos de configuración. La adaptación consiste en hacer opcional el fichero mapping model [13] a la hora de generar un producto. Una vez adaptado, se procede a la creación de un paquete de Python que se pueda instalar en cualquier entorno de desarrollo. como resultado, tenemos el paquete UVEngine [20][14]

[13]Mapping model:Capítulo 5

[14]El paquete es empleado en el proyecto UVEngine-Resolver y en los entornos de desarrollo de las plantillas.

4.7.4. Desarrollo aplicación Frontend Angular

En esta fase se desarrolla la aplicación web que permite la interacción con el usuario y la visualización de los archivos de configuración generados. Como hemos mencionado anteriormente, consta de un proyecto Angular presente en el siguiente repositorio GitHub [25]. La idea fundamental es permitir al usuario elegir entre un abanico de plantillas y versiones para que pueda decidir que producto generar en base a la configuración que se aporte como entrada. Debe funcionar como una abstracción de los servicios de Backend para permitir al usuario trabajar desde una interfaz web. Angular utiliza un enfoque basado en componentes para la construcción de aplicaciones web, los componentes más relevantes son:

- `Common Components/` contiene los componentes que se emplean en toda la aplicación.
- `Pages/` contiene los componentes que representan las páginas de la aplicación.
- `Services/` contiene los servicios que se emplean en la aplicación.
- `Shared Components/` contiene los componentes que se emplean específicamente para la integración con el editor de mocano.
- `Structure/` contiene los componentes que representan la estructura de la aplicación.

No olvidemos que gran parte de la lógica de la aplicación se encuentra desarrollada como servicios de la aplicación bajo el directorio `Service/`:

- `Services/` contiene los servicios que se emplean en la aplicación, como manejar las peticiones a backend y reescribir por pantalla los resultados.

4.7.5. Desarrollo de los servicios de backend

En esta fase se desarrollan los servicios de backend que permiten la comunicación entre la aplicación web y la aplicación de UVEngine y los repositorios. El primer servidor de backend que vamos a generar será un servidor para gestionar las peticiones sobre las plantillas, a este servicio lo llamaremos `Repository Manager`. Consiste en un servicio Flask que traduce las peticiones a la API de GitHub y las devuelve a la aplicación web. Desde este servicio tendremos acceso a la lista de plantillas y la lista de versiones. Además, aportamos al siguiente

servicio un método para la descarga de las plantillas. El siguiente servicio es el encargado de resolver la variabilidad de una configuración y generar un producto, a este servicio lo llamaremos UVEngine-Resolver. Este servicio se encarga de la generación de productos a partir de una configuración válida; para ello se apoya en el paquete UVEngine que hemos desarrollado en la fase anterior.

4.7.6. Elaboración de la guía de uso

En esta fase se elabora una guía de uso que explica cómo utilizar la aplicación y las funcionalidades disponibles. Se elabora una guía detallada que describe cómo utilizar la aplicación, incluyendo instrucciones paso a paso, capturas de pantalla y ejemplos.

4.7.7. Elaboración de un manual de instalación

En esta sección se crea un manual de instalación que explica cómo instalar y configurar la aplicación en un entorno local.

4.7.8. Elaboración de un manual de desarrollo de plantillas

En esta parte se crea un manual de desarrollo de plantillas que explica cómo crear y mantener plantillas en el repositorio[15].

4.8. Elaboración de la memoria

En esta fase se redacta la memoria o informe final del proyecto, que documenta todo el proceso de desarrollo, desde la planificación hasta la implementación y las lecciones aprendidas. La memoria incluirá el análisis de los resultados, conclusiones y recomendaciones para trabajos futuros.

[15]https://github.com/M4rdom/Templates

5

Modelado de las plantillas

Este espacio pretende describir de forma detallada el modelado de las plantillas y a qué necesidad responden en la construcción de un fichero de configuración.

5.1. Introducción

Dar cabida a la gran cantidad de escenarios y posibles casos de uso para la aplicación se ha descrito anteriormente como una tarea abrumadora. Aunque pudiéramos crear una plantilla lo suficientemente expresiva para dar cabida a todos los escenarios de uso existentes hasta la fecha, las plantillas quedarían desfasadas conforme se suceden cambios en las metodologías, surgen nuevas imágenes base, otras quedan desactualizadas, etc. La obra resultante llegaría a ser impracticable en cuanto a la cantidad de opciones y configuraciones que se podrían dar. El objetivo del desarrollo de todas las plantillas de este proyecto no pretende llegar a un nivel extremadamente avanzado de personalización, sino que sirvan como base para la creación de plantillas más específicas y detalladas.

5.2. Componentes de las plantillas

5.2.1. Feature model

El archivo .uvl es el archivo principal de la plantilla, contiene la descripción de las características y las restricciones de la plantilla. Este archivo lo usará la extensión UVLS [4] para generar una configuración válida que cumpla con las restricciones textuales que dictemos en el archivo, de este modo nos aseguramos de que estamos aplicando una configuración válida.

```
features
    Sandwich
        mandatory
            Bread
        optional
            Sauce
                alternative
                    Ketchup
                    Mustard
            Cheese
constraints
    Ketchup => Cheese
```

Figura 8: Representación de un Feature model en lenguaje UVL [6].

5.2.2. Plantillas Jinja

Las plantillas Jinja son los archivos de plantilla que se utilizan para generar los productos a partir de una configuración válida del modelo. La idea es exponer estas plantillas ante una configuración ya validada usando UVLS [4]. Estas plantillas constituyen el núcleo del funcionamiento, en ellas se declaran de forma explícita las opciones de configuración y las variables que se pueden modificar.

```
{%- if Compiled_Version_Tag is defined %}
FROM mongo:{{Compiled_Version_Tag}}
{%- elif Compiled_Version_Digest is defined %}
FROM mongo@{{Compiled_Version_Digest}}
{%- else %}
FROM mongo:latest
{%- endif %}



{%- if MONGO_INITDB_ROOT_USERNAME is defined %}

# Set the password for the default postgres user DO NOT USE IN PRODUCTION
ENV MONGO_INITDB_ROOT_USERNAME=postgres
{%- endif %}
{%- if MONGO_INITDB_ROOT_PASSWORD is defined %}

# Set the password for the default postgres user DO NOT USE IN PRODUCTION
ENV MONGO_INITDB_ROOT_PASSWORD=postgres
{% endif %}
```

Figura 9: Ejemplo Plantilla para la configuración de un servicio de base de Datos para MongoDB.

Las plantillas tienen que tener la estructura necesaria de modo que, usando la Sintaxis de Jinja, el motor de plantillas sea capaz de crear un producto funcional para una configuración válida. La imagen de arriba puede ser parte de un solo componente de la plantilla. Dividir las plantillas en componentes[16] nos permite estructurar la plantilla; esto puede verse con un ejemplo en la Subsección Plantilla Dockerfile.

[16]Los componentes de una plantilla no son más que importar otros ficheros .jinja para dar estructura y evitar manejar un archivo gigante que recojan todas la opciones de configuración.

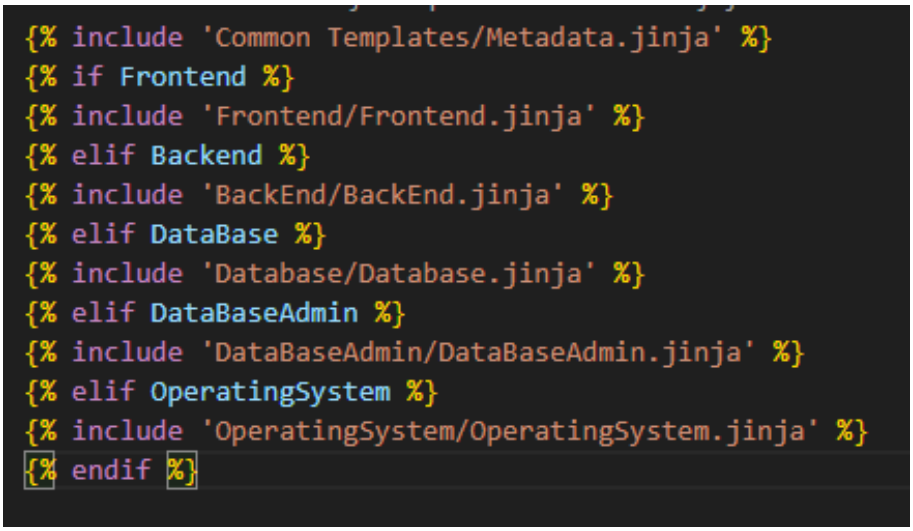


Figura 10: Ejemplo de división por componentes.

5.2.3. Mapping Model

Es el único componente completamente opcional a la hora de resolver la variabilidad, tampoco es necesario crearlo para el desarrollo de las plantillas.

Contiene la estructura de un fichero csv de tres columnas. Las primeras columnas permiten mapear features sean estas booleanas o no, a otros features Booleanas que podrán ser empleadas en las plantillas Jinja, la tercera columna permite especificar una feature. Este archivo permite renombrar features y hacerlas más comprensibles para las plantillas Jinja. También es el responsable de hacer funcionar configuraciones sobre otras plantillas.

```
# WebServer Version Parameters
Custom_WebServer_Frontend_Version, WebServer_Version_Tag,
WebServer_Frontend_Version_Tag, WebServer_Version_Tag
WebServer_Frontend_Version_Digest, WebServer_Version_Digest,

# Frontend WebServer
Apache_WebServer_Frontend, Apache
Nginx_WebServer_Frontend, Nginx
Node_WebServer_Frontend, Node

# WSGI Gunicorn Parameters
parameter, value
interface_Gunicorn, interface
workers_Gunicorn, workers
module_Gunicorn, module
callable_Gunicorn, callable

# WSGI uWSGI Parameters
interface_uWSGI, interface
workers_uWSGI, workers
module_uWSGI, module
callable_uWSGI, callable
```

Figura 11: Ejemplo Mapping Model.

5.3. Plantilla Dockerfile

Como hemos visto, un fichero Dockefile es un archivo de texto que contiene una serie de instrucciones que Docker [11] utilizará para construir una imagen de contenedor.

5.3.1. Feature model

La plantilla Dockerfile cuenta con el modelo más grande del proyecto. La idea para crear estos modelos de plantilla es subdividir el modelo en partes más pequeñas y manejables, de forma que se pueda trabajar con ellas de forma independiente. El criterio a la hora de hacer subdivisiones es similar al criterio que usan repositorios como Docker Hub [9] que divide sus imágenes por categorías dependiendo de su función. En este contexto podemos construir una primera subdivisión que represente el resultado final esperado, es decir, la primera feature será decidir si nuestra imagen tiene el objetivo de desplegar un contenedor que albergue un servicio de Frontend, Backend, Base de datos, etc.

Cada una de estas categorías contendrá a su vez una serie de subcategorías que representen

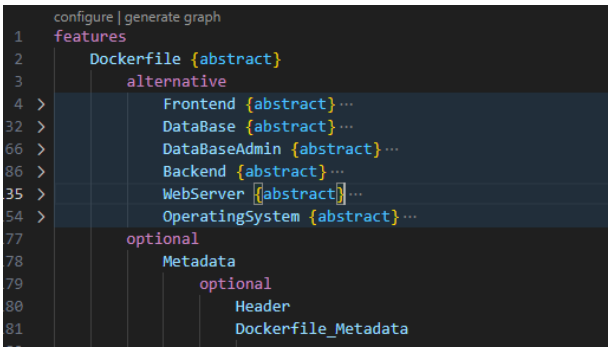


Figura 12: Captura de la primera subdivisión para Dockerfile.

las opciones de configuración que se pueden dar en cada una de ellas.

Centrémonos en el caso de uso de un Frontend. Comúnmente, al desplegar una imagen de frontend se sigue una estrategia de multi etapa [10] donde en una primera etapa compilamos la aplicación y en una posterior etapa la alojamos en un servidor web, esto con el objetivo de crear una imagen lo más pequeña posible al no contener ninguno de los ficheros fuente.

Figura 13: Captura de las opciones de configuración para frontend.

En este caso las subcategorías podrían ser el lenguaje, el puerto en el que se desplegará la aplicación, el comando que se ejecutará al iniciar el contenedor, etc. Estos son elementos ya mencionados en la sección de herramientas disponibles, como pueden ser las acciones del comando docker init en Docker Desktop [8].

Nuestro modelo permite recoger estas características además de otras, como pueden ser el

servidor web (Nginx, Apache, ...) que queremos junto con su versión, Podremos seleccionar el framework de desarrollo que se ha empleado, la versión en la que compilará la aplicación, etc. Las posibilidades van mucho más allá de lo que se recoge en esta versión del fichero Dockerfile, pero estas opciones ya expanden enormemente la capacidad de configuración que tienen otras herramientas.

5.3.2. Plantillas Jinja2

Una vez que contamos con el modelo de la plantilla, podemos proceder a la creación de las plantillas Jinja2 que son quienes realmente aportan el contenido al producto.

```
1  # ------------------ Build Stage ----------------------#
2
3  {% include 'Frontend/Framework/Framework.jinja'%}
4
5  # ------------------ Production Stage -----------------#
6
7  {% include 'Frontend/WebServer/WebServer.jinja'%}
8
9  {%- if Port is defined %}
10
11 EXPOSE {{Port}}
12 {% else %}
13
14 EXPOSE 80
15 {% endif %}
16
```

Figura 14: Captura Sección de la Plantilla Dockerfile. Podemos ver cómo la instrucción EXPOSE aparece en la plantilla Dockerfile a la hora de generar un producto con una configuración para exponer un servicio de frontend.

La sintaxis de Jinja2 nos permite aplicar cierta lógica a la hora de generar los ficheros de configuración, por ejemplo, podemos emplear condicionales para que ciertas partes del fichero se generen o no en función de la configuración que se haya dado.

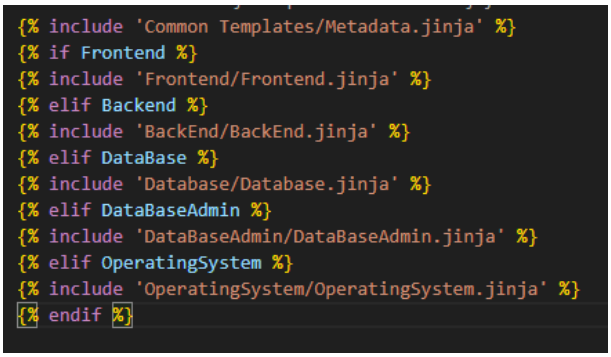


Figura 15: Captura de la primera subdivisión de la plantilla Dockerfile.

Dividir las plantillas no solo es útil porque las hace más manejables y fáciles de mantener, sino que también permite la reutilización de las mismas. Por ejemplo, si tenemos una plantilla que se repite en varias plantillas, podemos extraerla a un fichero aparte y reutilizarla en todas las plantillas que lo necesiten. Para este caso podemos generar una plantilla para los metadatos y labels[17] de la imagen.

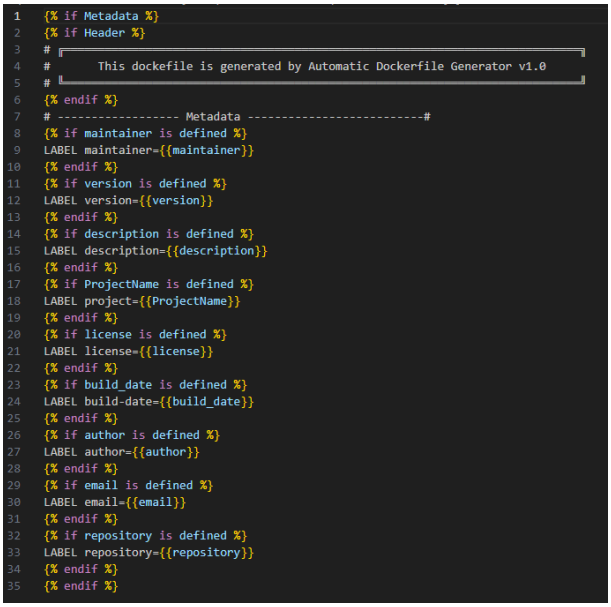


Figura 16: Captura de la primera subdivisión para Dockerfile.

[17] etiquetas

5.3.3. Mapping Model

El mapping model es un componente opcional que permite mapear features a otras features, esto es útil cuando queremos renombrar features o hacerlas más comprensibles para las plantillas Jinja. En este caso aplicamos dicha funcionalidad para paliar las restricciones actuales del parser existente del lenguaje UVL[18].

```
# WebServer Version Parameters
Custom_WebServer_Frontend_Version, WebServer_Version_Tag,
WebServer_Frontend_Version_Tag, WebServer_Version_Tag
WebServer_Frontend_Version_Digest, WebServer_Version_Digest,

# Frontend WebServer
Apache_WebServer_Frontend, Apache
Nginx_WebServer_Frontend, Nginx
Node_WebServer_Frontend, Node

# WSGI Gunicorn Parameters
parameter, value
interface_Gunicorn, interface
workers_Gunicorn, workers
module_Gunicorn, module
callable_Gunicorn, callable

# WSGI uWSGI Parameters
interface_uWSGI, interface
workers_uWSGI, workers
module_uWSGI, module
callable_uWSGI, callable
```

Figura 17: Captura de una sección del fichero Mapping Model plantilla Dockerfile.

[18]En UVL no podemos emplear nombres de features que contengan espacios o caracteres especiales o bien tampoco podemos repetir nombres de features en varias partes del modelo

5.3.4. Tests sobre la Plantilla Dockerfile

El diseño y construcción de las plantillas es un proceso incremental, debe comprobarse con regularidad si la intención a la hora de seguir desarrollando una plantilla es mantener una coherencia con lo añadido hasta el momento.

Con esta idea en mente, la solución propuesta consiste en la creación de pruebas que validen la correcta construcción de productos según la plantilla en la que estamos trabajando.

Estas pruebas deben ser capaces de comprobar que las plantillas generan un producto correcto. Para el caso del Dockerfile estas pruebas deben componerse de cada configuración y un producto esperado. Si nuestra plantilla es capaz de generar el mismo producto con la configuración dada de entrada, la prueba será considerada como exitosa.

Figura 18: Configuración Ejemplo 1 Backend.

```dockerfile
# Python Aplication Dockerfile
# Fask Aplication Dockerfile

# Avoid using latest tag in production
FROM python:latest

# Keeps Python from buffering stdout and stderr to avoid situations where
# the application crashes without emitting any logs due to buffering.
ENV PYTHONUNBUFFERED=1

# Create a non-privileged user that the app will run under.
# See https://docs.docker.com/go/dockerfile-user-best-practices/
ARG UID=10001
RUN adduser \
    --disabled-password \
    --gecos "" \
    --home "/nonexistent" \
    --shell "/sbin/nologin" \
    --no-create-home \
    --uid "${UID}" \
    appuser
# Install the application dependencies
RUN --mount=type=cache,target=/root/.cache/pip \
    python -m pip install --no-cache-dir -r requirements.txt

WORKDIR /app

COPY . /app
# Install Gunicorn This is not recommended for production, requirements.txt should be used instead

RUN pip install --no-cache-dir gunicorn

USER appuser

# Run the application using Gunicorn
CMD ["gunicorn", "-w", "4", "-b", "0.0.0.0:80", "holamundo.py:app"]

# Make port 5000 available to the world outside this container
EXPOSE 5000

```

Figura 19: Producto Esperado para la configuración de la Figura 18.

Es evidente que ciertos cambios más radicales plantean refactorizar algunos test. Aun así, es aconsejable validar el diseño de los nuevos elementos de la plantilla solucionando estas incoherencias. En el contexto de esta plantilla Dockerfile las pruebas se han realizado empleado las librerías de pytest [7] para la generación de los productos esperados. Estos test pueden encontrarse en el repositorio para el desarrollo de la plantilla dockerifle [**Dockerfile_template_dev**].

5.3.5. Resto de Plantillas

El proceso de creación de plantillas es similar para el resto de plantillas. La diferencia radica en la complejidad de las plantillas y en la cantidad de opciones de configuración que se pueden dar. Estas plantillas pueden encontrarse en el repositorio [26].

6

Arquitectura y Descripción del sistema

Esta sección describe la arquitectura y el modelado del sistema, incluyendo los componentes, la estructura y el diseño de la aplicación resultante en una imagen.

6.1. Visión general de la arquitectura

La arquitectura de la aplicación se divide en contenedores para manejar cada uno de los servicios de los que hace uso. La aplicación se divide en 4 servicios principales: Repository-Manager, UVEngine-Resolver, Frontend y una instancia de nginx que hace de reverse proxy. Podemos agrupar los contenedores de Repository-Manager y UVEngine-Resolver como el backend de nuestra aplicación. Nuestro backend es responsable de la generación de los archivos de configuración a partir de las plantillas y de la comunicación con el repositorio de las plantillas. Por otra parte, contamos con un servidor de nginx que ofrece nuestra aplicación Angular; a esta parte la llamaremos Frontend. El frontend es responsable de la interacción con el usuario y la visualización de los archivos de configuración generados. Como hemos mencionado anteriormente, también contamos con una instancia de Nginx que actúa como reverse proxy y que se encarga de enrutar las peticiones a los servicios de backend y frontend por comodidad en el desarrollo.

6.1.1. Diagrama de infraestructura

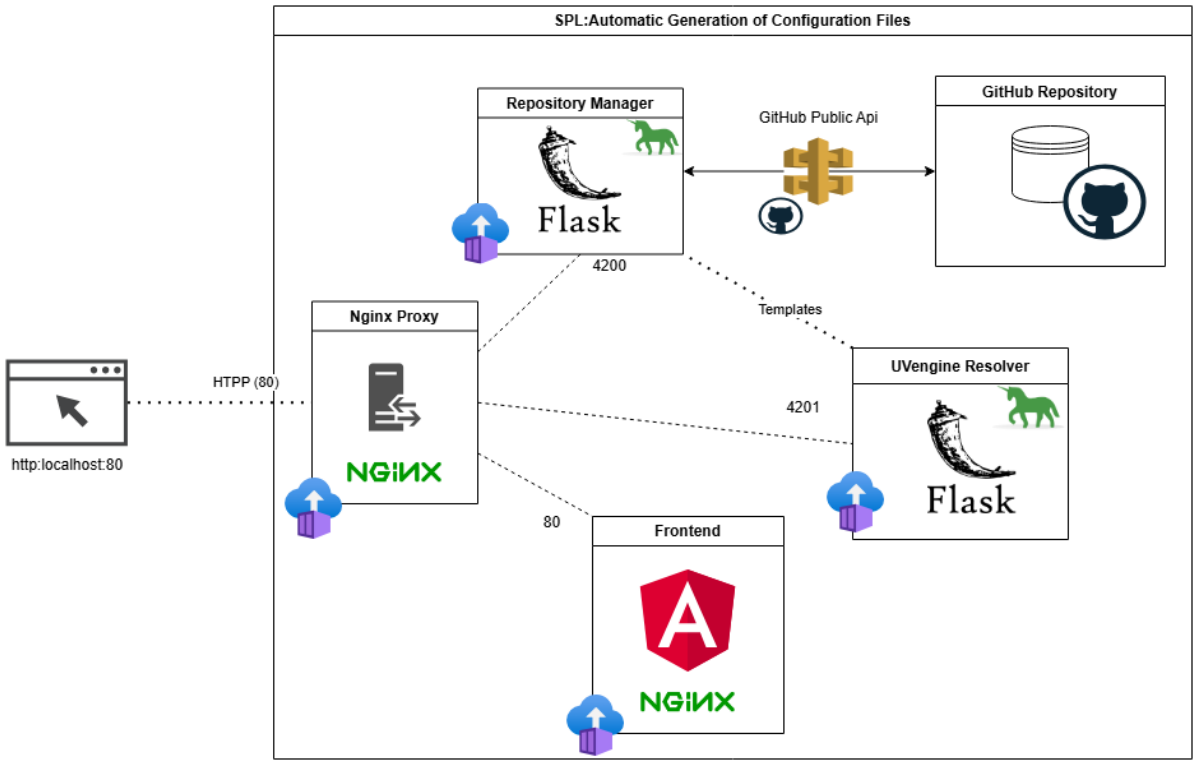


Figura 20: Diagrama infraestructura de Contenedores.

6.2. Descripción de servicios de backend

El backend de nuestra aplicación se ha dividido en dos servicios independientes con el objetivo de simplificar el desarrollo de la aplicación. Al contar con dos servicios independientes, podemos escalarlos de forma independiente y mantenerlos de forma independiente en un futuro si fuera necesario.

6.2.1. Repository-Manager

Este servicio se encarga de servir o traducir las peticiones que puede realizar el cliente o UVEngine-Resolver a la API de GitHub. Estas peticiones son obtener el archivo .uvl de la plantilla seleccionada, obtener el listado de plantillas disponibles y obtener el listado de versiones disponibles para cada plantilla, Esta aplicación está desarrollada en Python y emplea el framework Flask para la creación de la API, esta API se encarga de servir los archivos uvl de las plantillas y las versiones disponibles. Para hacer funcionar esta API se emplea la API pública de GitHub que permite acceder al contenido de repositorios. En este caso el repositorio al que queremos acceder es el repositorio Templates [26] que contiene las plantillas disponibles. También contamos con un Endpoint para descargar todo el contenido de las plantillas del repositorio. Este Endpoint es útil para mantener actualizadas las plantillas en el servidor UVengine Resolver.[19]

[19]En este momento es más eficiente descargar todo el repositorio de plantillas en una única petición para no abusar de la API pública de GitHub.

6.2.2. UVEngine-Resolver

El segundo servicio es UVEngine-Resolver, que se encarga de resolver la variabilidad a partir de recibir como parámetros un JSON que contiene el nombre de la plantilla, la versión a utilizar y la configuración. El resultado de la función será un JSON que contiene el producto generado a partir de la configuración dada.

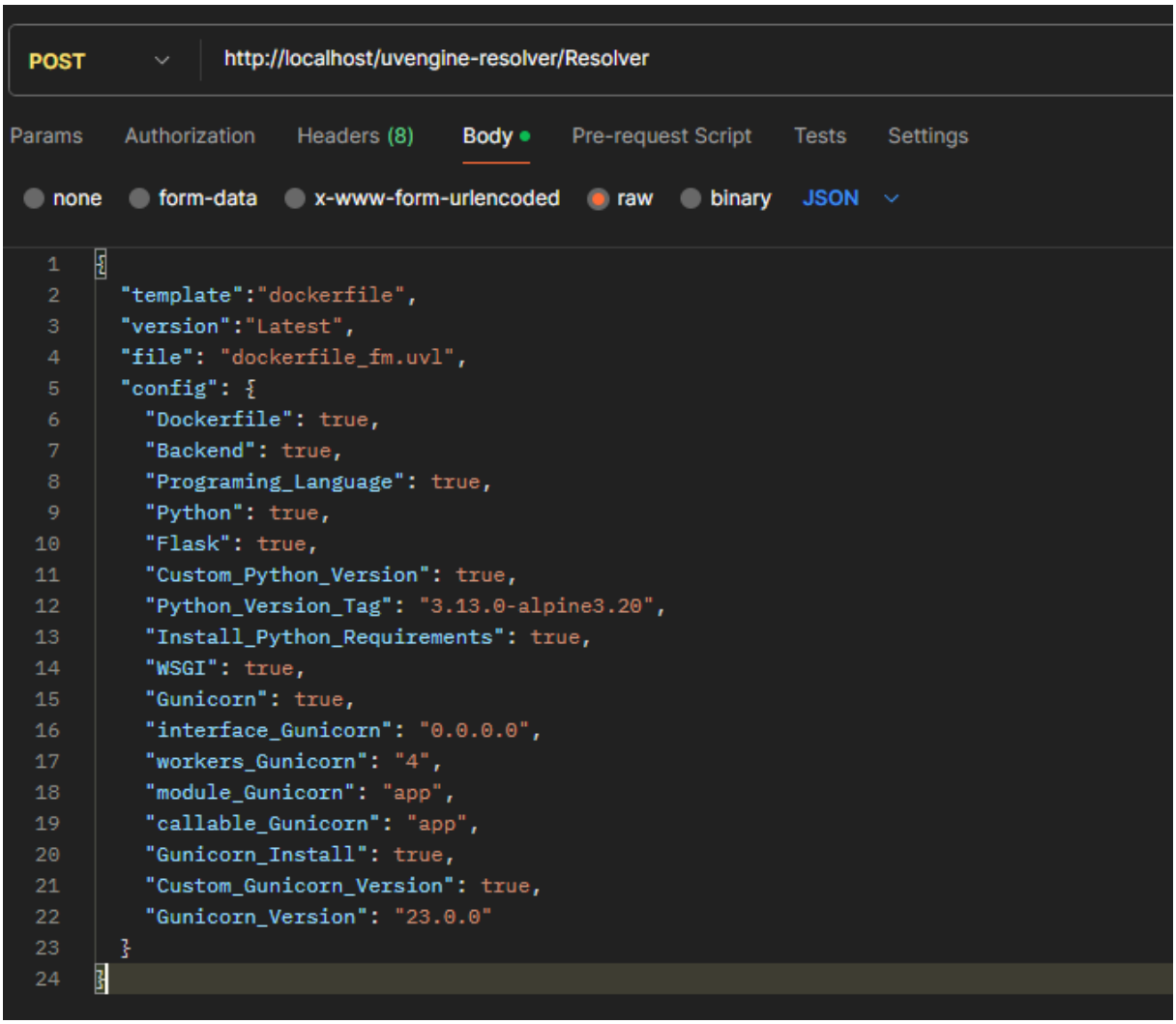


Figura 21: Ejemplo de una petición POST a UVEngine-Resolver.

También contamos con algunos Endpoints que pueden recibirse tanto desde el cliente como del Servicio Repository-Manager para actualizar las plantillas.

6.3. Descripción de Frontend

Como hemos comentado anteriormente, el Frontend es la parte de la aplicación que se encarga de la interacción con el usuario y la visualización de los archivos de configuración generados. Se trata de una aplicación web desarrollada en Angular que permite al usuario seleccionar una plantilla y una versión y generar un producto a partir de una configuración válida.

La aplicación queda alojada en un contenedor de Docker que se encarga de servir la aplicación web en el puerto 80. Este contenedor se comunica con el servicio Repository-Manager para obtener las plantillas y versiones disponibles a través de peticiones del cliente. Para servir la aplicación usamos el servidor web Nginx.

7
Extensibilidad

En esta sección se pretende hacer hincapié en la extensibilidad de la aplicación. Como se ha mencionado anteriormente, el proyecto tiene en mente que pueda adaptarse a las necesidades futuras.

Al almacenar las plantillas en un repositorio público de GitHub, permite a cualquier usuario modificar, añadir o eliminar plantillas de forma sencilla y ordenada. Esto permite que el desarrollo de plantillas se maneje de forma independiente al resto de la aplicación y que se puedan añadir nuevas funcionalidades sin necesidad de modificar el código de la aplicación.

7.1. Añadir Plantillas

Existiendo varias formas en las que se pudiera haber diseñado la forma con la que añadir nuevas plantillas a la aplicación aprovechando características de GitHub, se ha optado por la solución más sencilla y directa. Añadir una nueva plantilla es tan sencillo como añadir una nueva carpeta al repositorio de plantillas de GitHub, Esta carpeta debe contener una estructura de directorios específica que contenga los archivos necesarios para la plantilla, como son el Feature Model en UVL, las plantillas Jinja2 y el archivo de Mapping Model, este último archivo es opcional. Existen instrucciones detalladas de cómo añadir una plantilla al repositorio de plantillas.[20] o bien en el Apéndice Manual de Desarrollo.

[20]Repositorio de plantillas: `https://github.com/M4rdom/Templates`.

7.1.1. Ampliar la Plantilla Dockerfile

La plantilla Dockerfile recoge alguna de las características más interesantes que ya cubren otras herramientas, pero queda claro que todavía existe mucho margen de mejora en cuanto a la personalización de las opciones de configuración.

La plantilla no es a día de hoy capaz de cubrir todas las opciones de configuración que se pueden dar en un fichero Dockerfile, pero tiene el potencial de ser ampliada y mejorada en el futuro. Ampliar esta plantilla es sencillo, se cuenta ya con un repositorio de desarrollo para la plantilla[21] que contiene la plantilla y los test necesarios para validarla. Una vez que se haya ampliado la plantilla, se puede añadir al repositorio de plantillas de GitHub y utilizarla para generar productos.

Al seleccionar una feature que no ha sido desarrollada, el producto resultante (en este caso el fichero Dockerfile) contendrá el siguiente mensaje de error:

Figura 22: Mensaje de Error en el Producto Final.

[21] Repositorio para el desarrollo de la Plantilla Dockerfile `https://github.com/M4rdom/Dockerfile-Template-Dev` [**Dockerfile_template_dev**].

8

Despliegue de la aplicación

8.1. Despliegue local

Para desplegar el proyecto y todas sus capas se hace uso de la herramienta Docker, que permite la creación de contenedores ligeros y portables que pueden ejecutarse en cualquier entorno. Para el despliegue local se hace uso de Docker Compose [8], que permite definir y ejecutar aplicaciones Docker multicontenedor generando todos los servicios necesarios para el despliegue de la aplicación. Esto supone una abstracción de la infraestructura necesaria para el despliegue de la aplicación, permitiendo la creación de un entorno de desarrollo y pruebas aislado y reproducible. En el proyecto se encuentra el fichero docker-compose.yml que contiene la configuración necesaria para el despliegue de la aplicación en un entorno local.

```
version: '3.8'

services:
  fronted:
 build:
   context: ./Frontend
   Dockerfile: Dockerfile
 container_name: frontend
 ports:
   - "4200:80"
```

```
 repository-manager:
build:
  context: ./Repository-Manager
  Dockerfile: Dockerfile
container_name: repository-manager
ports:
  - "5000:5000"

 uvengine-resolver:
build:
  context: ./UVEngine-Resolver
  Dockerfile: Dockerfile
container_name: uvengine-resolver
ports:
  - "5001:5001"
depends_on:
  - repository-manager

 nginx-reverse-proxy:
build:
  context: ./Nginx-Reverse-Proxy
  Dockerfile: Dockerfile
container_name: nginx-reverse-proxy
ports:
  - "80:80"
```

Como se observa, el código se compone de 4 servicios:

- Frontend: Servicio que se encarga de servir la aplicación web.
- Repository Manager: Servicio que se encarga de servir los archivos UVL de las plantillas y las versiones disponibles.

- UVEngine Resolver: Servicio que se encarga de resolver la variabilidad a partir de recibir como parámetros un fichero JSON que contiene el nombre de la plantilla, la versión a utilizar y la configuración.
- Nginx Reverse Proxy: Servicio que se encarga de redirigir las peticiones de la aplicación web al servidor de backend y frontend.

La instrucción dockercompse up creará los contenedores en la configuración ya escrita. La web será accesible en la dirección localhost en el puerto 80.

9 Resultados

Esta parte del documento se centra en los resultados obtenidos durante el desarrollo del proyecto, incluyendo los resultados de las pruebas, la viabilidad del proyecto y la escalabilidad.

9.1. Resultados Obtenidos

Cumpliendo los objetivos propuestos al inicio del proyecto, se ha desarrollado una aplicación que permite la generación automática de archivos de configuración a partir de plantillas.

Un resumen de los resultados del proyecto es el siguiente:

- La plataforma web de la aplicación aporta una forma ágil de crear productos software a partir de ellas.
- Las plantillas para generar ficheros como Dockerfiles recogen un interesante número de posibilidades, entre las que se encuentran las opciones más usadas. Una vez generamos una configuración, podemos simplemente descargar el fichero generado o copiarlo al portapapeles y quedaría listo para su uso.
- El empleo de la extensión UVLS para generar una configuración puede ser muy eficiente a la hora de generar productos con opciones de configuración relacionadas entre sí. Además, tenemos la posibilidad de editar el producto final antes de utilizarlo.
- Las plantillas existentes aportan unas funcionalidades interesantes que pueden explotarse a la hora de generar productos como ficheros de configuración.
- El repositorio es fácilmente ampliable para abarcar nuevas plantillas y desarrollar nuevas funcionalidades.

Resumen de los desarrollos resultantes del proyecto:

- Se ha desarrollado una aplicación web que permite la interacción con el usuario y la visualización de los archivos de configuración generados.
- Se adaptado el uso del proyecto UVEngine que permite la generación automática de archivos de configuración a partir de plantillas, adaptándola a las necesidades del proyecto.
- Se ha desarrollado una aplicación de Repository-Manager que permite la comunicación con el repositorio de plantillas de GitHub. [26] .
- Se ha desarrollado una aplicación de UVEngine-Resolver que permite la resolución de la variabilidad a partir de una configuración.
- Se ha creado la plantilla para la generación de ficheros Dockerfile a partir de una configuración, además de un entorno de desarrollo para la misma.
- Se ha desarrollado plantillas auxiliares para ficheros de configuración Nginx, dockerignore, etc.
- Se han desarrollado pruebas para validar la correcta construcción de los productos a partir de las plantillas Dockerfile.
- Se ha aportado todo el código fuente y la documentación necesaria para la replicación del proyecto.
- Se ha generado una Memoria que documenta todo el proceso de desarrollo.
- Se han aportado apéndices para el uso más fundamental de la aplicación.

9.2. Viabilidad del proyecto

Como hemos recogido arriba, el proyecto es potencialmente viable, pero presenta algunas carencias que pueden ser corregidas. Es posible que el uso obligatorio de la extensión UVLS para generar los ficheros de configuración sea un punto en contra para la viabilidad del proyecto, El uso de la herramienta de configuración puede no estar a la altura en cuanto a comodidad de uso de otras herramientas de generación automática de archivos. Las soluciones actualmente disponibles son más cómodas al detectar las features del proyecto de forma autónoma, sin

necesidad de intervención del usuario. Esto supone que esta solución quede atrás en cuanto a comodidad. Es conveniente adaptar este sistema para que sea capaz de detectar algunas de las features de forma autónoma y dejar otras para la personalización del usuario, al menos en cuanto nos referimos a la generación del fichero Dockerfile.

9.3. Escalabilidad

Teniendo en cuenta que la escalabilidad de cualquier aplicación depende de la capacidad de sus servicios de generar nuevas instancias es en un principio viable, escalar horizontalmente cada uno de los servicios. Sin embargo, la escalabilidad de la aplicación se ve limitada actualmente por el uso de API pública de GitHub [16] para el servicio de Repository-manager visto en el Capítulo 6 que limita el número de peticiones que se pueden hacer en un periodo de tiempo.[22] En el escenario en el que se quiera llevar la aplicación a un entorno de producción, debe sortearse este problema bien por medio del uso de tokens de autenticación. Existe la posibilidad de que el usuario aporte un token personal de una cuenta de GitHub a la hora de desplegar el servicio o bien emplear un token propio desde GitHub.

En el escenario de superar dicho límite, desde la aplicación no se podrá descargar modelos UVL ni se podrá recargar el listado de plantillas ni versiones de las mismas. No obstante, se podrá resolver la variabilidad de las plantillas que ya se han descargado y se podrá generar productos a partir de ellas si se encuentran disponibles en el servidor de backend, gracias al UVEngine-Resolver.

[22] Actualmente el número de peticiones que puede realizarse a la API son 80 peticiones por minuto; esto es propenso a que cambien en un futuro.

10

Conclusiones y líneas Futuras

10.1. Conclusiones del Desarrollo del proyecto

El desarrollo del proyecto ha permitido alcanzar los objetivos planteados inicialmente. Se ha logrado implementar una solución eficiente y potencialmente escalable que cumple con los requisitos funcionales y no funcionales establecidos. Además, se ha adquirido un conocimiento profundo sobre las tecnologías utilizadas y se han identificado áreas de mejora para futuros desarrollos. El desarrollo de las plantillas ha sido conservador y no se ha considerado explorar más que casos básicos de uso. Esto era algo esperado si asumimos la gran casuística que tenemos. Sin embargo, se ha demostrado que la aplicación es capaz de generar productos válidos a partir de configuraciones válidas. Dado el nivel de desarrollo de las herramientas (UVLS) y tecnologías de las que hace uso este trabajo se ha demostrado que es posible llevar a cabo un proyecto de estas características.

10.2. Conclusión Personal

A nivel personal, el desarrollo de este trabajo me ha servido para introducirme de lleno en la tecnología de contenedores, cómo esta tecnología se aplica en servicios y qué implicaciones existen a nivel de rendimiento y seguridad. Considero que el proyecto tiene de base una serie de fortalezas que otra serie de herramientas no consigue ofrecer actualmente. Es posible que un proyecto más ambicioso y con más recursos pueda explotar estas fortalezas y llevar el diseño de productos por este camino. Sin duda, el proyecto queda lejos de tener aplicaciones funcionales reales, pero muestra un camino que puede ser explotado si se considera.

10.3. Aplicaciones Prácticas

Además de contar con una forma interesante de generar Dockerfiles, pueden existir dinámicas para aprovechar configuraciones válidas para generar otros ficheros necesarios, como, por ejemplo, podemos generar ficheros de configuración para el servidor de Nginx [30] directamente a partir de la configuración para generar un Dockerfile. Esto es gracias a los ficheros Mapping Models y a un diseño inteligente de las plantillas.

10.4. Desarrollo de nuevas plantillas

El proyecto está preparado para recibir nuevas plantillas manteniendo las existentes, por ejemplo pueden agregarse todo tipo de plantillas que requieran de la salida de un fichero de configuración, como pueden ser en el caso de herramientas como Ansible [39]. Una vez añadidas plantillas al repositorio, éstas estarán disponibles de la misma forma en la que están disponibles las plantillas existentes y solo será necesario recargar el listado de plantillas que se encuentra en la barra de navegación de la aplicación web.

10.5. Continuación en el desarrollo de plantillas ya existentes

Las plantillas existentes tienen un gran margen de mejora. En la situación actual no soportan la gran mayoría de las opciones de configuración que se pueden dar en un fichero Dockerfile. Además, las opciones que se manejan son propensas a quedar obsoletas y se necesita de una continua revisión de las plantillas en busca de problemas de diseño o seguridad. Estos son problemas que padecen otras herramientas que nos permiten generar el ficheros Dockerfile de forma automática.

10.6. Mejorar la experiencia del usuario

La generación de la configuraciones está ligada al desarrollo de la extensión UVLS. Dicha extensión permite al usuario generar una configuración válida desplegando el árbol de características, pero para ello el usuario necesita contar con VSCode y la extensión instalada para ser capaz de generar configuraciones válidas. La propuesta para este escenario sería la integración de dicha funcionalidad dentro de la plataforma web, La extensión UVLS es un proyecto

de código abierto y su integración en la aplicación web podría ser un proyecto en sí mismo.

En el desarrollo del trabajo se ha explorado esta posibilidad ejecutando dicha extensión en la versión web del editor VS Code, pero el resultado en cuanto a rendimiento hace que esta opción no sea viable.

La propia aplicación web puede ser mejorada en cuanto a la experiencia del usuario, añadiendo más información al usuario para colocar más contexto a la hora de seleccionar una plantilla y las posibilidades de combinar configuraciones. Otras opciones que pueden explorarse serían llevar la aplicación a un entorno de escritorio empleando Electron [12].

Otras mejoras que supondrían una gran aportación serían desarrollar herramientas que permitan la generación de configuraciones de forma automática reconociendo las necesidades del proyecto, como hacen otras aplicaciones como Starter [41].

10.7. Mejorar la escalabilidad de la aplicación

La escalabilidad de la aplicación se ve limitada por el uso de la API pública de GitHub para el servicio de Repository-manager. Por tanto, adaptar la aplicación a peticiones autorizadas a la API de GitHub es una mejora necesaria para llevar la aplicación a un entorno de producción.

Referencias

[1] Amazon Web Services. *The Difference Between Docker Images and Containers*. Accessed: 2024-10-21. 2024. URL: https://aws.amazon.com/es/compare/the-difference-between-docker-images-and-containers/#:~:text=Una%20imagen%20de%20Docker%2C%20o,el%20contenedor%20necesita%20para%20ejecutarse..

[2] Angular. *Angular - The modern web developer's platform*. Accedido: 22 de octubre de 2024. 2024. URL: https://angular.dev/.

[3] Angular. *Angular Material*. Accedido: 22 de octubre de 2024. 2024. URL: https://material.angular.io/.

[4] Caradhras. *UVLS - Universal Variability Language Support*. Accedido: 22 de octubre de 2024. 2024. URL: https://marketplace.visualstudio.com/items?itemName=caradhras.uvls-code.

[5] Visual Studio Code. *Developing inside a Container*. Accedido: 21 de octubre de 2024. 2024. URL: https://code.visualstudio.com/docs/containers/overview.

[6] UVL Community. *Universal Variability Language (UVL)*. Accedido: 21 de octubre de 2024. 2024. URL: https://universal-variability-language.github.io/.

[7] pytest Development Team. *Pytest Documentation*. Accedido: 22 de octubre de 2024. 2024. URL: https://docs.pytest.org/en/stable/.

[8] Docker. *Docker Desktop*. Accedido: 21 de octubre de 2024. 2024. URL: https://www.docker.com/products/docker-desktop/.

[9] Inc. Docker. *Docker Hub*. Accedido: 22 de octubre de 2024. 2024. URL: https://hub.docker.com/.

[10] Inc. Docker. *Use multi-stage builds*. Accedido: 22 de octubre de 2024. 2024. URL: https://docs.docker.com/build/building/multi-stage/.

[11] Docker, Inc. *Docker: Accelerate How You Build, Share, and Run Modern Applications*. Accessed: 2024-10-21. 2024. URL: https://www.docker.com/.

[12] Electron. *Electron: Desarrolla aplicaciones de escritorio con JavaScript, HTML y CSS*. Accedido: 22 de octubre de 2024. 2024. URL: https://www.electronjs.org/es/.

[13] Python Software Foundation. *Python*. Accedido: 22 de octubre de 2024. 2024. URL: https://www.python.org/.

[14] GitHub. *GitHub Personal Access Tokens*. Accedido: 22 de octubre de 2024. 2024. URL: https://github.com/settings/tokens.

[15] Inc. GitHub. *GitHub*. Accedido: 22 de octubre de 2024. 2024. URL: https://github.com/.

[16] Inc. GitHub. *GitHub REST API Documentation*. Accedido: 22 de octubre de 2024. 2024. URL: https://docs.github.com/en/rest?apiVersion=2022-11-28.

[17] Google Cloud. *What are Containers?* Accessed: 2024-10-21. 2024. URL: https://cloud.google.com/learn/what-are-containers?hl=es.

[18] Gunicorn. *Gunicorn*. Accedido: 22 de octubre de 2024. 2024. URL: https://gunicorn.org/.

[19] José Manuel Horcas. *UVEngine*. Accedido: 22 de octubre de 2024. 2024. URL: https://github.com/jmhorcas/uvengine.

[20] José Manuel Horcas. *UVEngine 1.0.0*. Accedido: 22 de octubre de 2024. 2024. URL: https://pypi.org/project/uvengine/1.0.0/.

[21] IBM. *Orquestación de contenedores*. Accedido: 21 de octubre de 2024. 2024. URL: https://www.ibm.com/es-es/topics/container-orchestration.

[22] KeepCoding. *¿Qué es el archivo .dockerignore?* Accedido: 22 de octubre de 2024. 2024. URL: https://keepcoding.io/blog/que-es-archivo-dockerignore/.

[23] Panamerican Latam. *¿Qué es un prompt en IA y para qué sirve?* Accedido: 21 de octubre de 2024. 2024. URL: https://panamericanlatam.com/que-es-un-prompt-en-ia-y-para-que-sirve/.

[24] M4rdom. *Automatic Generation of Configuration Files for Deploying Services in Docker*. Accedido: 22 de octubre de 2024. 2024. URL: https://github.com/M4rdom/Automatic-generation-of-configuration-files-for-deploying-services-in-Docker.

[25] M4rdom. *Frontend*. Accedido: 22 de octubre de 2024. 2024. URL: https://github.com/M4rdom/Frontend.

[26] M4rdom. *Templates*. Accedido: 22 de octubre de 2024. 2024. URL: https://github.com/M4rdom/Templates.

[27] O'Reilly Media. *Docker Certified Associate*. Accedido: 22 de octubre de 2024. 2024. URL: https://www.oreilly.com/library/view/docker-certified-associate/9781839211898/c5ecd7bc-b7ed-4303-89a8-e487c6a220ed.xhtml#uuid-1a5da664-fb76-4e56-bdb0-83255dde9e78.

[28] Microsoft. *Monaco Editor*. Accedido: 22 de octubre de 2024. 2024. URL: https://microsoft.github.io/monaco-editor/.

[29] Microsoft. *Visual Studio Code*. Accedido: 22 de octubre de 2024. 2024. URL: https://code.visualstudio.com/.

[30] Inc. Nginx. *Nginx*. Accedido: 22 de octubre de 2024. 2024. URL: https://nginx.org/en/.

[31] Node.js. *Node.js*. Accedido: 22 de octubre de 2024. 2024. URL: https://nodejs.org/en/.

[32] Inc. npm. *ngx-json-viewer*. Accedido: 22 de octubre de 2024. 2024. URL: https://www.npmjs.com/package/ngx-json-viewer.

[33] Inc. npm. *npm*. Accedido: 22 de octubre de 2024. 2024. URL: https://www.npmjs.com/.

[34] OpenAI. *ChatGPT*. Accedido: 21 de octubre de 2024. 2024. URL: https://chatgpt.com/.

[35] Podman Project. *Podman: A Tool for Managing Containers and Pods*. Accessed: 2024-10-21. 2024. URL: https://podman.io/.

[36] Pallets Projects. *Flask Documentation (3.0.x)*. Accedido: 22 de octubre de 2024. 2024. URL: https://flask.palletsprojects.com/en/3.0.x/.

[37] Pallets Projects. *Jinja Documentation (3.1.x)*. Accedido: 22 de octubre de 2024. 2024. URL: https://jinja.palletsprojects.com/en/3.1.x/.

[38] PyPI. *Python Package Index*. Accedido: 22 de octubre de 2024. 2024. URL: https://pypi.org/.

[39] Inc. Red Hat. *Ansible Automation Platform*. Accedido: 22 de octubre de 2024. 2024. URL: https://www.ansible.com/.

[40] Git SCM. *Git*. Accedido: 22 de octubre de 2024. 2024. URL: https://git-scm.com/.

[41] Starter. *Starter: Herramienta de desarrollo*. Accedido: 22 de octubre de 2024. 2024. URL: https://ww1.startwithdocker.com/?usid=15&utid=31190731855.

[42] *UVL Playground*. Accedido: 22 de octubre de 2024. 2024. URL: https://uvl.uni-ulm.de/.

[43] Wikipedia. *Feature Model*. Accedido: 21 de octubre de 2024. 2024. URL: https://en.wikipedia.org/wiki/Feature_model.

Apéndice A
Manual de Instalación

A.1. Requisitos Previos

Para lanzar la aplicación en un entorno local, debemos contar con los siguientes requisitos previos:

- Navegador Web
- Docker
- Visual Studio Code con la extensión UVLS [4] instalada.
- Git

A.2. Descargar el código fuente

Para descargar el código fuente de la aplicación, debemos clonar el repositorio de GitHub [24][23] en un directorio local. usando la instrucción `git clone` o bien descargar el repositorio en formato zip.

A.3. Despliegue del Servicio

Para desplegar el servicio debemos situarnos en la raíz del proyecto y ejecutar el siguiente comando:

```
docker-compose up
```

[23]https://github.com/M4rdom/Automatic-generation-of-configuration-files-for-deploying-services-in-Docker

A.4. Acceso a la aplicación

Una vez que los servicios se hayan desplegado correctamente, podremos acceder a la aplicación web en la dirección localhost en el puerto 80.

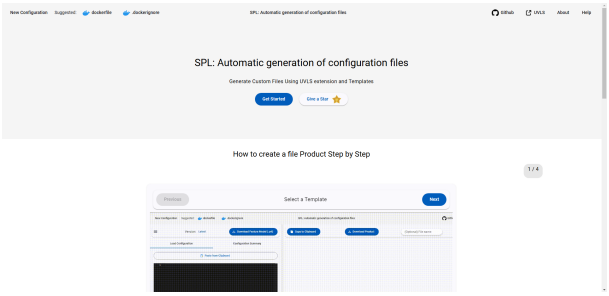


Figura 23: Página Principal

A.4.1. GitHub API

Debemos recordar que la aplicación hace uso de la API pública de GitHub [16] para el servicio de Repository-manager, por lo que es posible que se alcance el límite de peticiones permitidas por la API. Para superar este límite se puede emplear un token de autenticación de GitHub [14] para aumentar el número de peticiones permitidas. Para un uso estándar no es necesario emplear un token de autenticación.

A.5. Terminar el servicio

Para terminar el servicio debemos ejecutar el siguiente comando en la raíz del proyecto:

```
docker-compose down
```

Apéndice B
Manual de Uso

B.1. Inicio de la aplicación

Para iniciar la aplicación debemos acceder a la dirección localhost en el puerto 80.

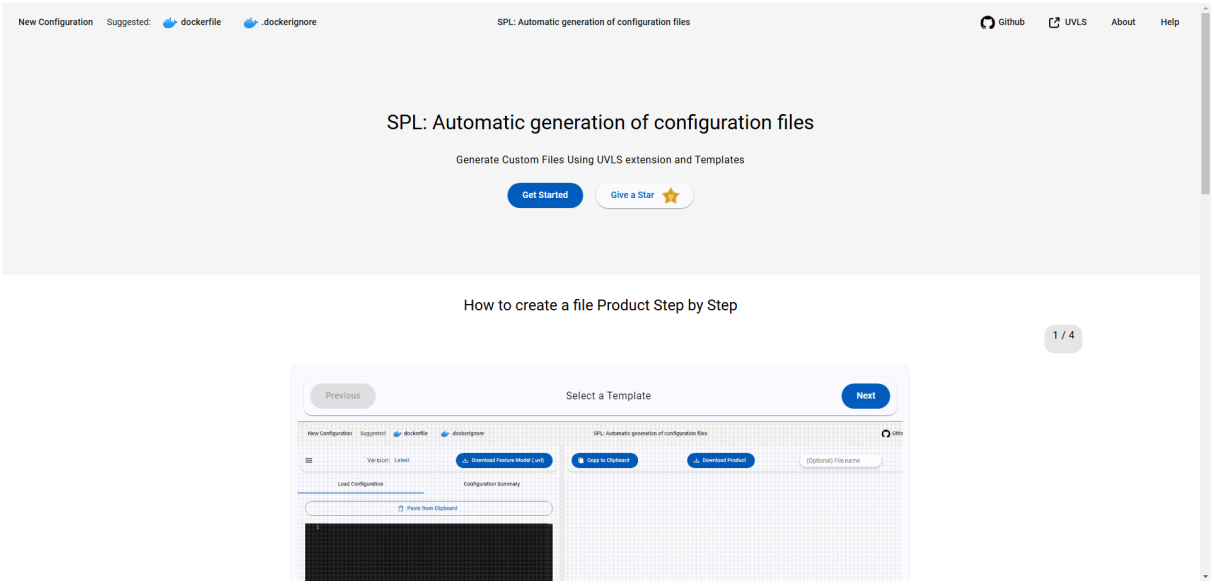


Figura 24: Página Principal.

B.2. Barra de Navegación

La barra de navegación de la aplicación web contiene las siguientes opciones:

Figura 25: Página Principal.

- New Configuration: Permite al usuario seleccionar una plantilla y una versión y generar un producto a partir de una configuración válida.
- Dockerfile: Permite al usuario crear un Dockerfile a partir de una configuración válida.

- .dockerignore: Permite al usuario crear un fichero .dockerignore a partir de una configuración válida.
- SPL: Automatic Generation of Configuration Files for Deploying Services in Docker: Permite al usuario acceder a la página principal.
- GitHub: Link a la página de GitHub del proyecto.
- UVLS: Link a la extensión UVLS en el Visual Studio Code Marketplace.
- About: Página con información sobre el proyecto.
- Help: Página con información sobre cómo usar la aplicación.

B.3. Seleccionar plantilla

Para seleccionar una plantilla debemos acceder a la página New Configuration y seleccionar una plantilla de la lista desplegable.

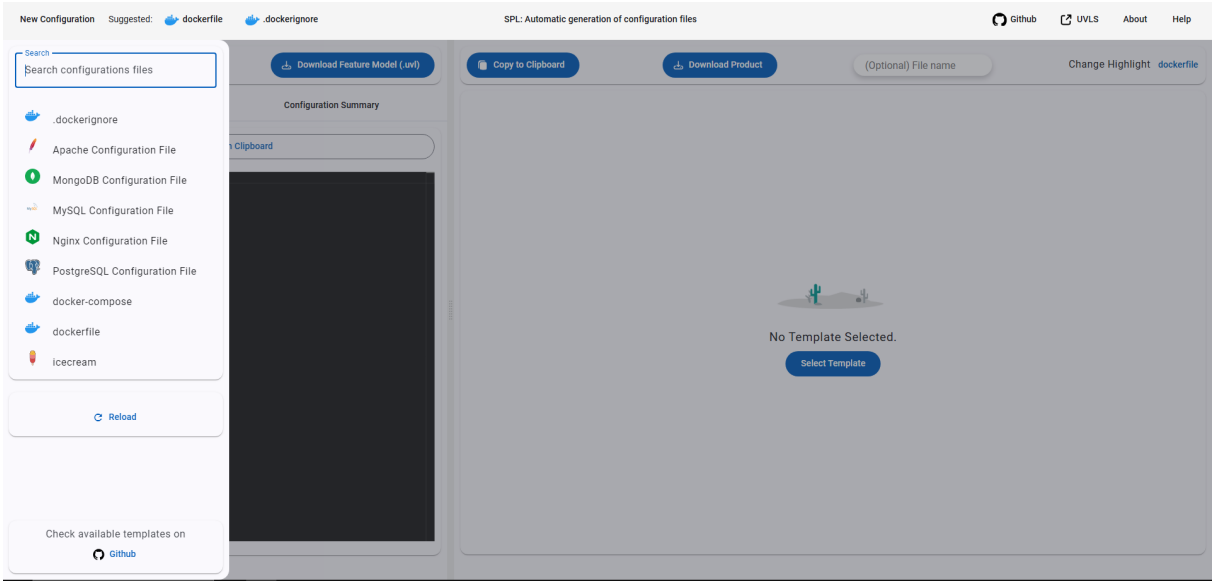


Figura 26: Barra Desplegable Selector de Plantillas.

B.3.1. Seleccionar versión de la plantilla

Para seleccionar la versión de la plantilla, abrimos el desplegable Version; por defecto tiene el valor de Latest.

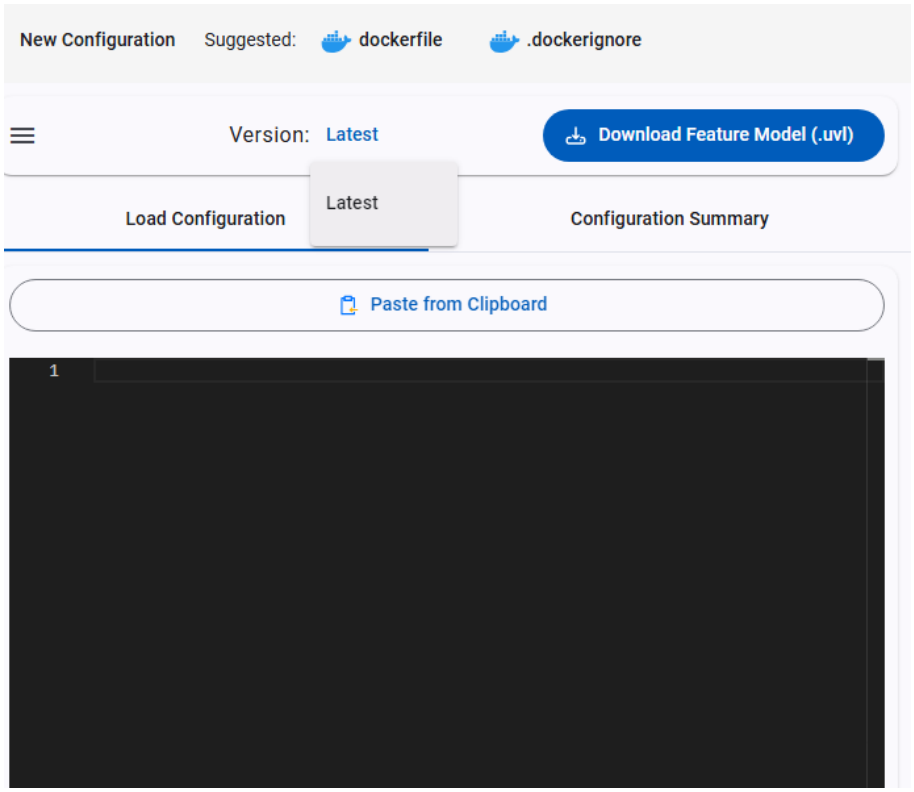


Figura 27: Captura Versiones Disponibles.

24

B.3.2. Refrescar Plantillas

En la página de arriba tendremos la opción de actualizar la lista de plantillas disponibles haciendo clic en el botón Reload.

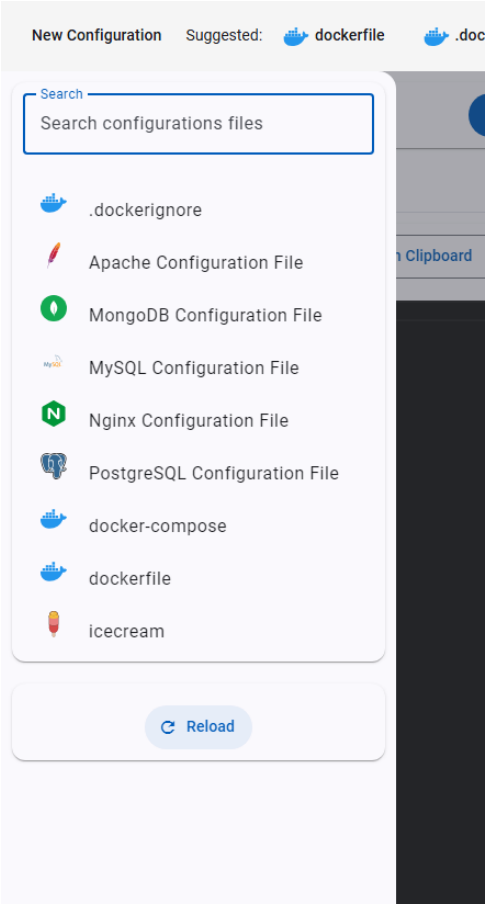


Figura 28: Captura Versiones Disponibles.

B.3.3. Cargar configuración

Para cargar una configuración debemos hacer clic en el botón Load Configuration y copiar en el editor un fichero de configuración valido Opcionalmente, podemos pegar la configuración desde el portapapeles.

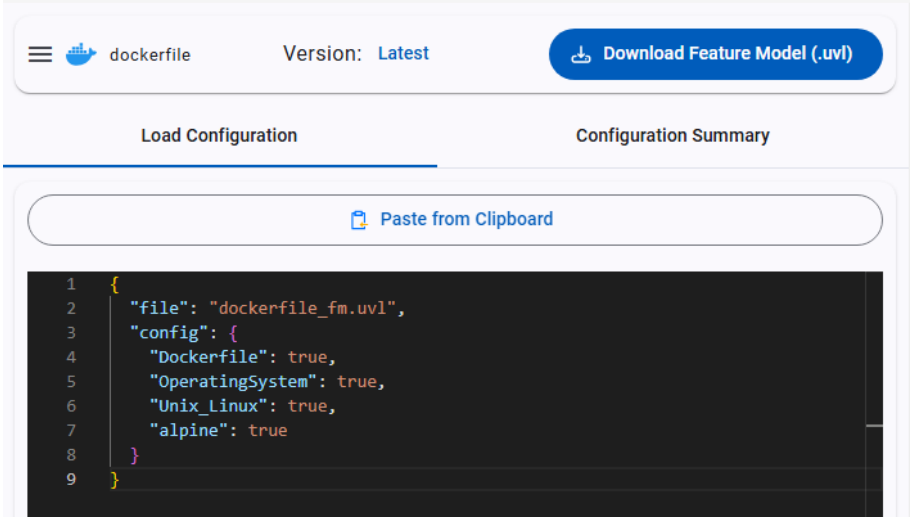


Figura 29: Editor para cargar configuración,

B.3.4. Descargar Fichero Generado

Una vez que hemos cargado la configuración y seleccionado una plantilla y una versión, podemos descargar el fichero generado haciendo clic en el botón Download File y opcionalmente añadiendo un nombre al fichero de salida. O bien simplemente copiamos al portapapeles.

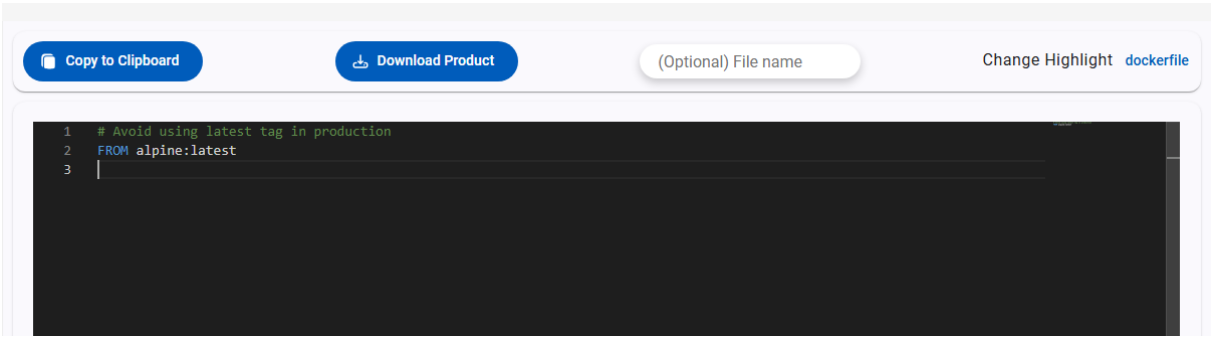


Figura 30: Editor para guardar Configuración.

B.3.5. About Page

About Page contiene información sobre el proyecto, el motivo de su existencia, los objetivos que pretende alcanzar y su desarrollador.

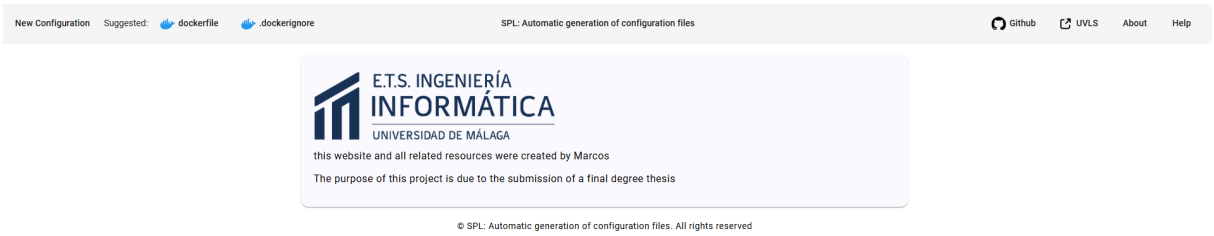


Figura 31: Página Principal.

B.3.6. Help Page

Help Page contiene información sobre la terminología del proyecto.

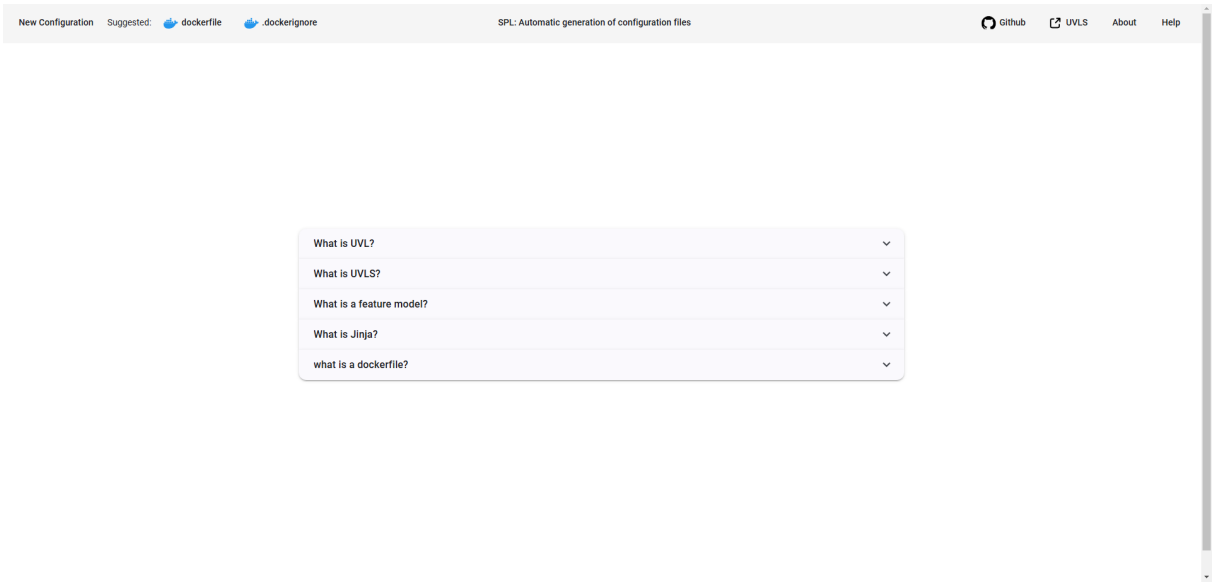


Figura 32: Captura Página de ayuda.

Apéndice C
Manual Desarrollo de Plantillas

C.1. Introducción

Este manual describe los pasos necesarios tanto para añadir nuevas plantillas al repositorio de plantillas [26] como para modificar las plantillas existentes. una vez añadidas las plantillas al repositorio, será necesario actualizar la lista de plantillas dentro de la aplicación véase el Manual de Uso.

C.1.1. Como añadir nuevas plantillas al repositorio

Para añadir una nueva plantilla al repositorio de plantillas debemos seguir los siguientes pasos:

- Generar la Estructura de directorios apropiada para la plantilla:
  - Crear una carpeta con el nombre de la plantilla.
  - Dentro de la Carpeta, Crear una carpeta con el nombre de la versión.
  - Dentro del nuevo Directorio, Crear las Carpetas Feature Model, Jinja Templates y opcionalmente Mapping Model.
- Añadir el archivo con nombre ≤Nombre de la Plantilla>_fm.uvl"dentro del directorio Feauture Model.
- Añadir las plantillas Jinja a la carpeta Jinja Templates, la plantilla principal debe estar bajo el nombre ≤Nombre de la Plantilla>.jinja".
- (Opcional) Añadir el archivo de Mapping Model a la carpeta Mapping Model bajo el nombre ≤Nombre de la Plantilla>_mapping_model.csv".
- Subir la plantilla al repositorio de plantillas haciendo push al repositorio.

- Actualizar la lista de plantillas en la aplicación web.

C.1.2. Estructura de Directorios

A modo de resumen, la estructura de directorios de una plantilla debe ser la siguiente:

```
/Template_Name
└─ /Version_Name
   ├─ /Feature Model
   │  └─ Template_Name_fm.uvl
   ├─ Jinja Templates
   │  ├─ /More Jinja Templates (Optinal)
   │  └─ Template_Name.jinja
   └─ /Mapping Model (Optinal)
      └─ Template_Name_mapping_model.csv
```

C.1.3. Como modificar las plantillas del repositorio

En el caso de que queramos modificar una plantilla ya existente en el repositorio de plantillas, debemos seguir los siguientes pasos:

- Modificar los archivos de la plantilla que queremos modificar respetando la estructura y nombres de la plantilla.
- Solicitar Pull Request al repositorio de plantillas
- Actualizar la lista de plantillas en la aplicación web.

Es recomendable generar estas plantillas a partir de un repositorio de desarrollo [**Dockerfile_template_d** para asegurar que las plantillas generan productos válidos realizando pruebas. Una vez que tengamos una plantilla lista, es solo cuestión de subirla al repositorio respetando la estructura original modificando únicamente la única plantilla que se desea modificar. Estas Instrucciones pueden verse con más detalle en el repositorio de plantillas [26].

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA