



UNIVERSIDAD DE MÁLAGA



Grado en Ingeniería Informática

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



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 metodo para la generacion de ficheros de configuracion para la contruccion de contenedores software. El fin del proyecto es generar una herramienta que permita a los desarrolladores definir una configuracion y obtener un producto personalizado como puerder ser un fichero dockerfile. Utilizando plantillas podemos aplicar sin concimiento previo del desarrollador las practicas recomendadas para la construccion de estos ficheros. Para lograr este objetivo se parte de la construccion de plantillas empleando la sintaxis de jinja y el motor de resolucion de variabilidad uvengine para generar el producto. La herramienta proporcionada es una aplicacion web que permite a los desarrolladores seleccionar una plantilla y version de la misma para resolver la variabilidad y obtener un producto. Actualmente, los desarrolladores pueden definir la configuracion haciendo uso de la extension UVLS en vscode y seleccionar las caracteristicas 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 conclusion el esfuerzo realizado permite generar plantillas de ficheros de configuracion con un alto grado de personalizacion en comparacion con otros métodos existentes.

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

Índice

1. Introduccion	11
1.1. Motivación	11
1.2. Objetivo	12
1.3. Resultados Esperados	13
1.4. Estructura del documento	13
2. Estado del Arte	15
2.1. Contenedores	15
2.1.1. Porque se usan los contenedores	15
2.2. Imagenes	16
2.3. Docker	16
2.3.1. Dockerfile	16
2.3.2. dockerignore	16
2.3.3. Buenas Prácticas en la creacion de las imagenes	17
2.4. Problematica	19
2.5. Soluciones Actualmente Disponibles	19
2.5.1. Docker init	19
2.5.2. Docker Extension for Visual Studio Code	20
2.5.3. Inteligencia Artificial	21
2.6. Conclusiones Preliminares	22
2.7. Que se aporta en este contexto:	22
2.7.1. Software Product Line SPL	23
3. Tecnologías Empleadas	25
3.1. Universal Variability Language UVL	25
3.2. Docker	25
3.3. Node.js	26
3.4. Angular	26

3.4.1.	Angular Material	26
3.4.2.	Monaco Editor	27
3.4.3.	ngx-json-viewer	27
3.5.	Nginx	27
3.6.	Python	27
3.6.1.	Flask	28
3.6.2.	Gunicorn	28
3.6.3.	Pypi	28
3.7.	Jinja	28
3.7.1.	Jinja2	28
3.8.	Git	29
3.9.	GitHub	29
3.9.1.	GitHub Public Api	29
3.10.	Visual Studio Code	30
3.10.1.	UVLS extension	30
4.	Metodologia del trabajo	31
4.1.	Introduccion	31
4.2.	Metodología del Desarrollo	31
4.2.1.	Justificación	32
4.2.2.	Notas del Procedimiento	32
4.2.3.	Limitaciones	33
4.3.	Fases de Estudio	33
4.4.	Estudio del Estado del Arte :	33
4.5.	Investigación diseño de dockerfiles	33
4.6.	Desarrollo de Plantillas	34
4.7.	Fase Implementacion	34
4.7.1.	Requisitos de la aplicación	34
4.7.2.	Diseño de la aplicación	35
4.7.3.	Desarrollo del Paquete uvengine	35
4.7.4.	Desarrollo aplicacion Frontend Angular	36

4.7.5.	Desarrollo de los servicios de backend	36
4.7.6.	Elaboración de la guía de uso	37
4.7.7.	Elaboración de un manual de instalación	37
4.7.8.	Elaboración de un manual de desarrollo de plantillas	37
4.8.	Elaboración de la memoria	37
5.	Modelado de las plantillas	39
5.1.	Introducción	39
5.2.	Componentes de las plantillas	40
5.2.1.	Feature model	40
5.2.2.	Plantillas Jinja	41
5.2.3.	Mapping Model	42
5.3.	Plantilla dockerfile	43
5.3.1.	Feature model	43
5.3.2.	Plantillas Jinja2	45
5.3.3.	Mapping Model	47
5.3.4.	Tests sobre la Plantilla dockerfile	48
5.3.5.	Resto de Plantillas	50
6.	Arquitectura y Descripción del sistema	51
6.1.	Vision general de la arquitectura	51
6.1.1.	Diagrama de infraestructura	52
6.2.	Descripcion de servicios de backend	53
6.2.1.	Repository-Manager	53
6.2.2.	UVEngine-Resolver	54
6.3.	Descripcion de Frontend	55
7.	Extensibilidad	57
7.1.	Añadir Plantillas	57
7.1.1.	Ampliar la Plantilla Dockerfile	58
8.	Despliegue de la aplicación	59
8.1.	Despliegue local	59

9. Resultados	63
9.1. Resultados Obtenidos	63
9.2. Viabilidad del proyecto	64
9.3. Escalabilidad	65
10. Conclusiones y líneas Futuras	67
10.1. Conclusiones del Desarrollo del proyecto	67
10.2. Conclusion Personal	67
10.3. Aplicaciones Prácticas	68
10.4. Desarrollo de Nuevas plantillas	68
10.5. Continuación en el desarrollo de plantillas ya existentes	68
10.6. Mejorar la experiencia del usuario	68
10.7. Mejorar la escalabilidad de la aplicacion	69
Apéndice A. Manual de	
Instalación	75
A.1. Requisitos Previos	75
A.2. Descargar el codigo fuente	75
A.3. Despliegue del Servicio	75
A.4. Acceso a la aplicacion	76
A.4.1. Github API	76
A.5. Terminar el servicio	76
Apéndice B. Manual de Uso	77
B.1. Inicio de la aplicacion	77
B.2. Barra de Navegación	77
B.3. Seleccionar plantilla	78
B.3.1. Seleccionar version de la platilla	79
B.3.2. Refrescar Plantillas	80
B.3.3. Cargar configuracion	80
B.3.4. Descargar Fichero Generado	81
B.3.5. About Page	82

B.3.6. Help Page	82
Apéndice C. Manual Desarrollo de Plantillas	83
C.1. Introducción	83
C.1.1. Como añadir nuevas plantillas al repositorio	83
C.1.2. Estructura de Directorios	84
C.1.3. Como modificar las plantillas del repositorio	84

1

Introduccion

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`¹ y `.dockerignore`², que definen cómo se construye la imagen de un contenedor en tecnologicas como Docker³

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 informacion necesaria para la correcta configuracion de los servicios,

¹Dockerfile

²dockerignore

³Docker

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 en para la generacion de ficheros de configuracion, 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 covertura 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 implementacion no tendran el objetivo de alcanzar un estado de produccion sino de servir como base para futuras implementaciones y desarrollos orientados a ello

1.4. Estructura del documento

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

- El Capitulo 10 incluye las conclusiones del trabajo y las recomendaciones para trabajos futuros.
- El Apendice A Es un manual de Instalacion de la aplicacion usando contenedores Docker
- El Apendice B Es un manual de Uso de la aplicacion web
- El Apendice C Es una guia para el desarrollo de plantillas, nuevas y existentes.

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 las posibles problemáticas. 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. [20]

2.1.1. Porque 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. [20] Es una solución de fácil aplicación, alto rendimiento y extremadamente portable. ⁴

⁴Existen 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 contenedor⁵ pero ese es el escenario más común.⁶

2.3. Docker

Docker [14] es un proyecto de código abierto que 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 Docker es una plataforma de código abierto que permite a los desarrolladores crear, desplegar y ejecutar aplicaciones en contenedores.⁷

2.3.1. Dockerfile

El archivo Dockerfile [11] 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 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 [25] 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

⁵Existe la posibilidad de crear contenedores de otras formas como a partir de otros contenedores

⁶construir un contenedor a partir de una imagen lo hace reproducible

⁷Existen alternativas a Docker como Podman [40] pero Docker es la más popular y ampliamente utilizada en la actualidad.

2.3.3. Buenas Prácticas en la creacion de las imagenes

Para la creacion de los ficheros dockerfile es recomendable seguir una serie de buenas practicas para garantizar la eficiencia y la seguridad en el despliegue de servicios en Docker. Estas practicas se recojen en la documentacion oficial de Docker [12] En la documentacion se abordan una serie de recomendaciones para la creacion de imagenes 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 configuracion Multietapa:
 - Crear imágenes en varias etapas permiten dividir la construcción de la imagen, lo que puede mejorar el rendimiento
 - Esto ayuda a reducir el tamaño final de la imagen al incluir 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.

8

⁸Para más informacion: <https://docs.docker.com/build/building/best-practices/>

2.4. Problemática

La dificultad de crear imagenes reside en la naturaleza de la aplicacion que queremos llevar a un contenedor, existen una gran cantidad de escenarios donde si bien la tarea es sencilla, esta puede esconder una complejidad implicita 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 imagenes que se crean.

La calidad de las imagenes pueden marcar una diferencia que puede llegar a ser crítica para posteriores etapas en el desarrollo de aplicaciones como son la Orquestacion de contenedores [24].

2.5. Soluciones Actualmente Disponibles

En la actualidad hay disponibles varias soluciones para disminuir la complejidad implicita que reside en en la configuracion de dichos ficheros dockerfile. Existen más herramientas que las representadas a continuacion 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 configuracion. Las opciones de configuracion se limitan a elegir el lenguaje de programacion, puerto en el que despliega la aplicacion y que comando se ejecutara al iniciar el contenedor. Las plantillas son creadas y mantenidas por la comunidad

9

⁹En este proyecto se reutilizan y se recogen ideas de varias de las plantillas que incluye Docker Init

```

Welcome to the Docker Init CLI!

This utility will walk you through creating the following files with sensible defaults for your project:
- .dockerignore
- Dockerfile
- compose.yaml
- README.Docker.md

Let's get started!

? What application platform does your project use? [Use arrows to move, type to filter]
Go - suitable for a Go server application
> Python - suitable for a Python server application
Node - suitable for a Node server application
Rust - suitable for a Rust server application
ASP.NET Core - suitable for an ASP.NET Core application
PHP with Apache - suitable for a PHP web application
Java - suitable for a Java application that uses Maven and packages as an uber jar
Other - general purpose starting point for containerizing your application
Don't see something you need? Let us know!
Quit

```

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

2.5.2. Docker Extension for Visual Studio Code

La extension de Docker para visual estudio code [5] permite entre otras muchas de sus funcionalidades la generacion del fichero dockerfile entre otros. su comportamineto es similar al de Docker init, permitiendo entre otras opciones elegir el lenguaje de desarrollo del proyecto y que comando se ejecutara al iniciar el contenedor.

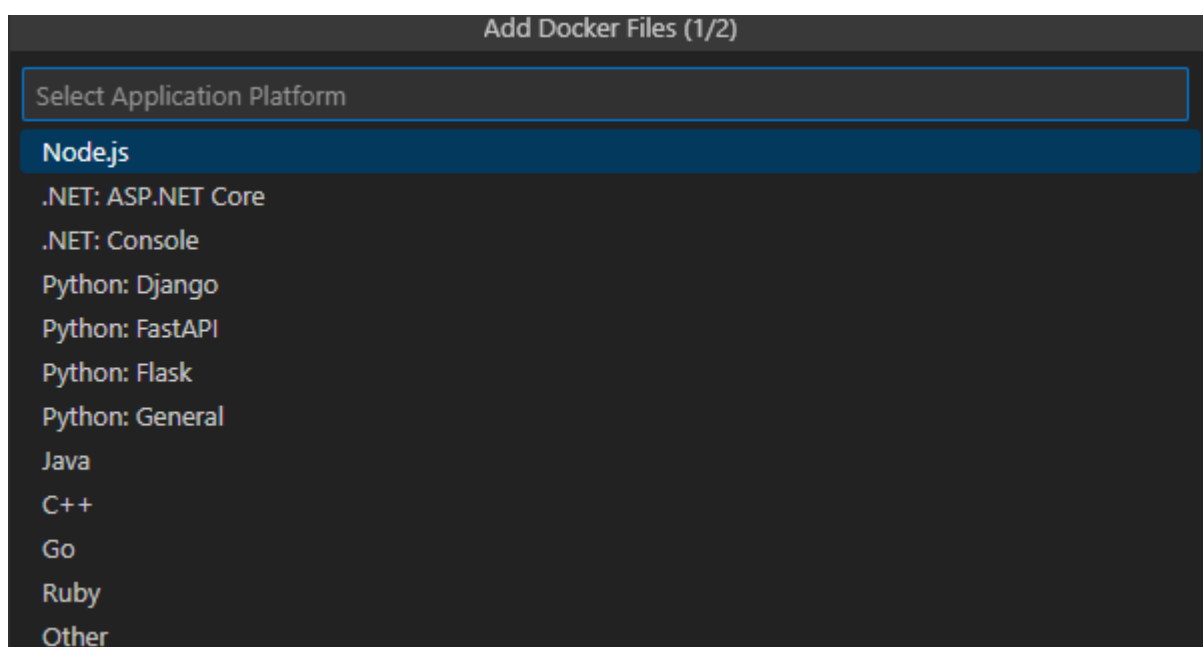


Figura 2: Plantillas Disponibles en la Extesnsion para Docker de Visual Studio Code.

```

1  # For more information, please refer to https://aka.ms/vscode-docker-python
2  FROM python:3-slim
3
4  EXPOSE 5002
5
6  # Keeps Python from generating .pyc files in the container
7  ENV PYTHONDONTWRITEBYTECODE=1
8
9  # Turns off buffering for easier container logging
10 ENV PYTHONUNBUFFERED=1
11
12 # Install pip requirements
13 COPY requirements.txt .
14 RUN python -m pip install -r requirements.txt
15
16 WORKDIR /app
17 COPY . /app
18
19 # Creates a non-root user with an explicit UID and adds permission to access the /app folder
20 # For more info, please refer to https://aka.ms/vscode-docker-python-configure-containers
21 RUN adduser -u 5678 --disabled-password --gecos "" appuser && chown -R appuser /app
22 USER appuser
23
24 # During debugging, this entry point will be overridden. For more information, please refer
25 CMD ["gunicorn", "--bind", "0.0.0.0:5002", "myapp.py:app"]
26 |

```

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 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 prompt¹⁰ de entrada. como magickpen [31]¹¹ o alternativas con propósito más general como chatgpt [39]¹²

¹⁰Instrucción o texto inicial proporcionado a una herramienta generativa de IA para dirigir la generación de respuestas o resultados específicos [26]

¹¹magickpen:<https://magickpen.com/templates/128/>

¹²chatgpt:<https://chat.openai.com/>

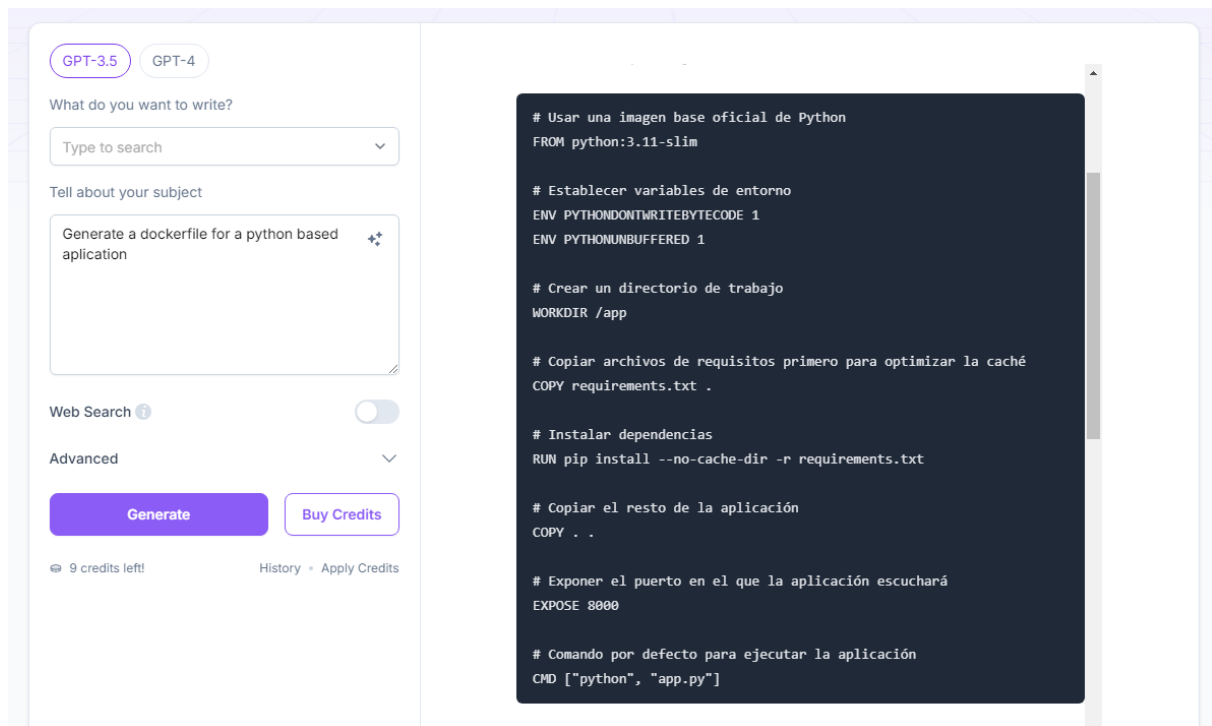


Figura 4: Captura de un resultado de magickpen

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 más final de las opciones de configuración. No obstante suponen un punto de partida para el desarrollo de aplicaciones

2.7. Que 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 deja de lado la variabilidad y las posibles opciones que existen a la hora de construir las imágenes, el proyecto proporcionará 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. Software Product Line SPL

En el proyecto tendremos que construir nuestras propias plantillas que formaran 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 un SPL, los sistemas se descomponen en características (features) que representan funcionalidades o comportamientos del sistema. El principal artefacto para modelar la variabilidad en SPL 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) [48] 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.

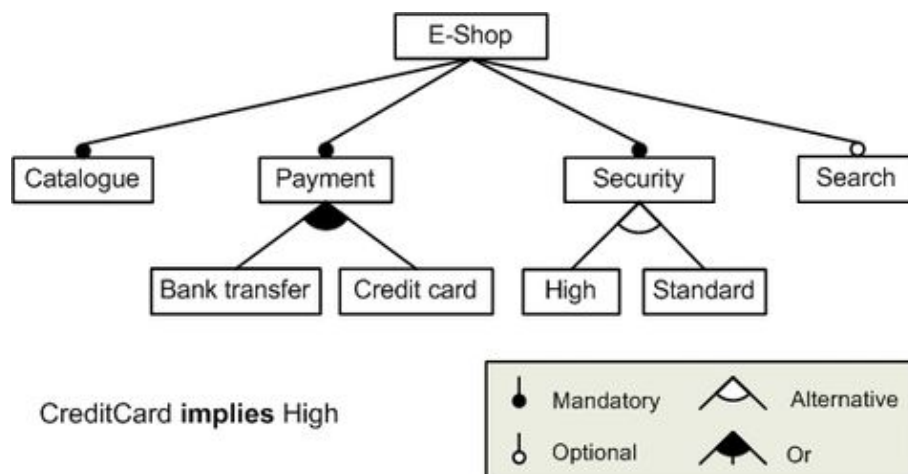


Figura 5: Ejemplo Representación Gráfica de un feature model [48]

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 de 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 validas.

3.2. Docker

La apliccion 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 utilizacion de Docker Desktop es complemento con la consola de comandos. Esta herremienta ayuda al proyecto en el momento en que lo hacer portable y facil de desplegar en cualquier entorno.

3.3. Node.js

[36] para el desarrollo de la aplicacion web se ha optado por la utilizacion de Node.js como entorno de ejecucion de JavaScript para el desarrollo Frontend. Esta herramienta constituye una plataforma de desarrollo en tiempo de ejecución cuenta con la ventaja de contar con una extensa biblioteca de módulos que simplifican el desarrollo de aplicaciones web. Con npm (Node Package Manager) [38], 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 version v22.10.0 y la version v10.9.0 del gestor de paquetes npm

3.4. Angular

[2] En cuanto al desarrollo de la aplicacion web se ha optado por la utilizacion de Angular como framework de desarrollo. El motivo de esta decision es la idea de emplear un framework moderno para emplear los metodos de desarrollo actuales y las mejores practicas en el desarrollo de aplicaciones web. Emplear otros framework de desarrollo como React o Vue.js tambien hubiera sido una opcion valida 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 requeria nuestra aplicacion ahorrando tiempo en la implementacion de las mismas. La version empleada es Angular v18¹³

3.4.1. Angular Material

[3] Para el diseño de la interfaz grafica de la aplicacion web se ha optado por la utilizacion los componentes provenientes de los modulos de Angular Material. Siguiendo la linea de diseño de Material Design, Angular Material proporciona una serie de componentes y directivas que facilitan la creacion de interfaces de usuario modernas y atractivas. Los componentes tienen una reaccion conocida por el usuario y son faciles de implementar en la aplicacion web.

14

¹³Angular 18 no es la ultima version disponible pero al depender de otros desarrollo se ha optado por la utilizacion de esta version

¹⁴La ultima version disponible es 18.2.9

3.4.2. Monaco Editor

[33] Para la edicion de los archivos de configuracion se ha optado por la utilizacion de Monaco Editor como editor de codigo. Monaco Editor es un editor de codigo fuente basado en la web que se utiliza en varios proyectos de Microsoft, como Visual Studio Code, Azure DevOps y el editor de codigo de GitHub. cuenta con una serie de características que lo hacen ideal para la edicion de archivos de configuracion, como la resaltado de sintaxis, el autocompletado y la verificacion de errores. tambien es altamente personalizable y extensible, lo que permite adaptarlo a las necesidades del proyecto.¹⁵

3.4.3. ngx-json-viewer

[37] Para la visualizacion de los archivos de configuracion se ha optado por la utilizacion 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 codigo abierto que se puede integrar facilmente en aplicaciones Angular y personalizar segun las necesidades del proyecto.

3.5. Nginx

[35] 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 configuracion que requiere para su correcto funcionamiento, ademas 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 aplicacion web al servidor de backend. y como servidor web para servir el contenido estatico de la aplicacion web.

3.6. Python

[16] 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

¹⁵La ultima version de angular en la que es compatible es la v18 por este motivo se ha desarrollado en esta version 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

[41] 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

[21] 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

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

3.7. Jinja

[42] 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 creacion de archivos de configuracion personalizados y adaptables a las necesidades del usuario. Todas las plantillas desarrolladas han usando la sintaxis de Jinja2

3.8. Git

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

3.9. GitHub

[18] 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, codigo ...

3.9.1. GitHub Public Api

[19] Para la gestion de las plantillas se ha empleado la API publica de GitHub para la obtencion de las plantillas disponibles. Esta desion 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 estan visibles y son accesibles y modificables por cualquier usuario de GitHub. Simplificando el proceso de creacion, mantenimiento y actualizacion de las plantillas.

3.10. Visual Studio Code

[34] Visual Estudio Code es un editor de código fuente desarrollado por Microsoft que se utiliza para escribir, editar y depurar código. Su mencion 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 de generador de ficheros de configuración de los que la aplicación hace uso.

3.10.1. UVLS extension

[4] UVLS es una extensión de Visual Studio Code que proporciona soporte para la Universal Variability Language en adelante 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 menor importancia empleadas en *VSCode* son:

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

4

Metodologia del trabajo

4.1. Introduccion

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

La investigacion y analisis tenia el objetivo de conocer en profundidad las tecnologias y herramientas empleadas para porstesiromete estudiar una posible aplicacion 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 planificacion de la implementacion de la aplicacion, para la implementacion de la aplicacion se ha optado por la utilizacion de una metodologia de desarrollo que permita la evolucion del proyecto a un producto minimo viable en el menor tiempo posible.

4.2. Metodología del Desarrollo

Para la implementacion 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 tecnologias empleadas. Los cambios más relevaten 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 aplicacion. La implementacion 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. Para adaptarnos a requerimientos no planeados y encajar nuevos ideas que pudieran sucederse. En todo momento el codigo es subcestible 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.

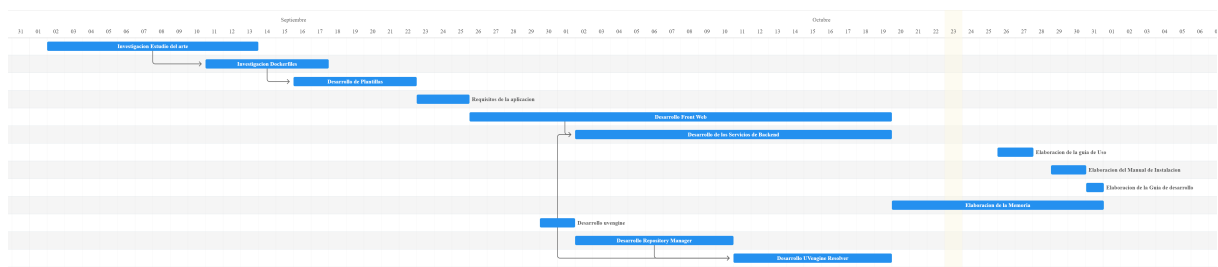


Figura 6: Diagrama de gantt

16

4.2.2. Notas del Procedimiento

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

¹⁶Diagrama generado por la herramienta <https://www.plandemejora.com/crear-diagrama-de-gantt-online/>

4.2.3. Limitaciones

El desarrollo de un proyecto de estas características requiere de una planificación y una metodología de trabajo adecuada. En este sentido, como se han identificado anteriormente una serie de limitaciones y restricciones que pueden afectar al desarrollo del proyecto. para el caso el desarrollo de las plantillas tiene que verse limitado en cuanto a formato y contenido, ya que no se puede abarcar todas las posibles opciones de configuracion.

El tiempo que se puede dedicar al proyecto tiene que estar dentro de unos limites 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 tecnologia de los contenedores fue exencial la lectura de Modern Infrastructures and Applications with Docker [32]
- Completar la intruduccin al lenguaje uvl con UVL Playground [47]

4.5. Investigación diseño de dockerfiles

Esta fase implica investigar y evaluar las diferentes opciones de configuracion 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 compredre la importacia de las buenas practicas en la creacion de imagenes fue necesario la lectura de la documentacion oficial de Docker Dockerfile reference [9]

4.6. Desarrollo de Plantillas

Esta fase implica la construcción el árbol de características y las restricciones textuales que orienten el espacio de configuraciones posibles del fichero Dockerfile. En este paso se pondrán en práctica todo el conocimiento adquirido en los pasos anteriores que tendrán como resultado 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 [11]. 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 un 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.

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 resume los componentes de la aplicación. una explicación más detallada puede encontrarse en el Capítulo 6

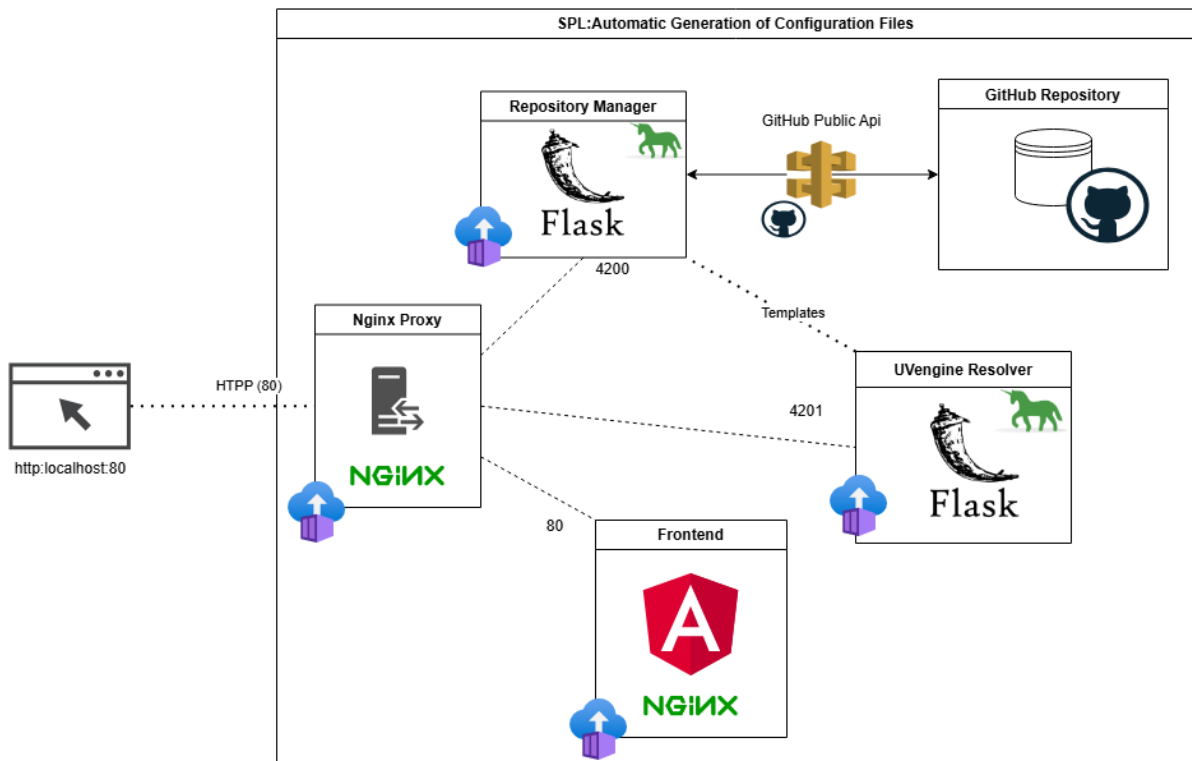


Figura 7: Diagrama de infraestructura

4.7.3. Desarrollo del Paquete uvengine

En esta fase se adapta el proyecto de la aplicación de uvengine [22], que permite la generación automática de archivos de configuración. La adaptación consiste en hacer opcional el fichero mapping model ¹⁷ 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 [23] ¹⁸

¹⁷Mapping model:Capítulo 5

¹⁸El paquete es empleado en el proyecto UVEngine-Resolver y en los entornos de desarrollo de las plantillas

4.7.4. Desarrollo aplicacion 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 [29] 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 configuracion que se aporte como entrada. Debe funcionar como una abstraccion 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 Compoenets/ contiene los componentes que se emplean en toda la aplicacion
- Pages/ contiene los componentes que representan las paginas de la aplicacion
- Services/ contiene los servicios que se emplean en la aplicacion
- Shared Components/ contiene los componentes que se emplean especificamente para la integracion con el editor de mocano
- Structure/ contiene los componentes que representan la estructura de la aplicacion

No olvidemos que gran parte de la logica de la aplicacion se encuentra desarrollada como servicios de la aplicacion bajo el directorio

- Services/ contiene los servicios que se emplean en la aplicacion, 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 sera un servidor para gestionar las peticiones sobre las plantillas, a este servio lo llamaremos Repository Manager. Consiste en un servicio flask que traduce las peticiones a la API de GitHub y las devuelve a la aplicacion web. Desde este servicio tendremos acceso a la lista de plantillas y la lista de versiones. Ademas aportamos al siguiente servicio

un metodo para la descarga de las plantillas. El siguiente servicio es el encargado de resolver la variabilidad de una configuracion y generar un producto, a este servicio lo llamaremos UVEngine-Resolver. Este servicio se encarga de la generacion de productos a partir de una configuracion valida, 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 seccion 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 ¹⁹

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.

¹⁹<https://github.com/M4rdom/Templates>

5

Modelado de las plantillas

Este espacio pretende describir de forma detallada el modelado de las plantillas y a que necesidad responden en la contruccion de un fichero de configuracion

5.1. Introducción

Dar cabida a la gran cantidad escenarios y posibles caso de uso para la aplicacion se ha descrito anteriormente como una tarea abrumadora. Aunque pudieramos crear una plantilla lo suficientemente expresiva para dar cabida a todos los escenarios de uso exitentes hasta la fecha, las plantillas quedarian desfasadas conforme se suceden cambios en las metodologias, surgen nuevas imagenes base, otras quedan desactualizadas etc. La obra resultante llegaria a ser impracticable en cuanto a la cantidad de opciones y configuraciones que se podrian dar. El objetivo del desarrollo todas las plantillas de este proyecto no pretende llegar a un nivel extremadamente avanzado de personalizacion, sino que sirvan como base para la creacion 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 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 configuracion valida o no del modelo. La idea es exponer estas plantillas ante una configuracion ya validada usando UVLS [4], Estas plantillas constituyen el nucleo del funcionamiento, en ella se declaran de forma explicita las opciones de configuracion y las variables que se pueden modificar.

```
1  {% if Compiled_Version_Tag is defined %}
2  FROM mongo:{{Compiled_Version_Tag}}
3  {% elif Compiled_Version_Digest is defined %}
4  FROM mongo@{{Compiled_Version_Digest}}
5  {% else %}
6  FROM mongo:latest
7  {% endif %}
8
9
10
11 {% if MONGO_INITDB_ROOT_USERNAME is defined %}
12
13 # Set the password for the default postgres user DO NOT USE IN PRODUCTION
14 ENV MONGO_INITDB_ROOT_USERNAME=postgres
15 {% endif %}
16 {% if MONGO_INITDB_ROOT_PASSWORD is defined %}
17
18 # Set the password for the default postgres user DO NOT USE IN PRODUCTION
19 ENV MONGO_INITDB_ROOT_PASSWORD=postgres
20 {% endif %}
```

Figura 9: Ejemplo Plantilla Para la configuracion 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 configuracion valida. La imagen de arriba puede ser parte de un solo componente de la plantilla. Dividir las plantillas en componentes ²⁰ nos permite estructurar la plantilla esto puede verse con un ejemplo en la subseccion Plantilla Dockerfile.

²⁰Los componentes de una plantilla no son más que importar otros archivos .jinja para dar estructura y evitar manejar un archivo gigante que recojan todas las opciones de configuracion

```

{% include 'Common Templates/Metadata.jinja' %}
{% if Frontend %}
{% include 'Frontend/Frontend.jinja' %}
{% elif Backend %}
{% include 'BackEnd/BackEnd.jinja' %}
{% elif DataBase %}
{% include 'Database/Database.jinja' %}
{% elif DataBaseAdmin %}
{% include 'DataBaseAdmin/DataBaseAdmin.jinja' %}
{% elif OperatingSystem %}
{% include 'OperatingSystem/OperatingSystem.jinja' %}
{% endif %}

```

Figura 10: Ejemplo de division por componentes

5.2.3. Mapping Model

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

Contiene la estructura de un fichero csv de tres con columnas. Las primeras columnas permiten mapear features sean estas booleanas o no a otros feature Booleanas que podran 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. Tambien 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 [14] 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 [10] donde divide sus imagenes por categorias dependiendo de su función. En este contexto podemos construir una primera subdivision que represente el resultado final esperado, es decir la primera feature sera 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 categorias contendra a su vez una serie de subcategorias que represen-

```

configure | generate graph
1  features
2    Dockerfile {abstract}
3      alternative
4 >      Frontend {abstract}...
32 >      DataBase {abstract}...
66 >      DataBaseAdmin {abstract}...
86 >      Backend {abstract}...
35 >      WebServer {abstract}...
54 >      OperatingSystem {abstract}...
77    optional
78      Metadata
79        optional
80          Header
81          Dockerfile_Metadata

```

Figura 12: Captura de la primera subdivision para dockerfile.

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

Centremonos en el caso de uso de un Frontend. Comunmente al desplegar una imagen de frontend se sigue una estrategia de multi etapa [13] donde en una primera etapa compilamos la aplicacion 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

```

Frontend {abstract}
  mandatory
    Framework {abstract}
      alternative
        Angular
        React
        Vue
      optional
        Custom_Compiled_Version
          alternative
            String Compiled_Version_Tag
            String Compiled_Version_Digest
        WebServer_Frontend {abstract}
          alternative
            Apache_WebServer_Frontend
              optional
                Custom_apache_config_file_Frontend
            Nginx_WebServer_Frontend
              optional
                Custom_nginx_config_file_Frontend
            NodeJs_WebServer_Frontend
              optional
                Custom_nodeJs_config_file_Frontend
          optional
            Custom_WebServer_Frontend_Version
              alternative
                String WebServer_Frontend_Version_Tag
                String WebServer_Frontend_Version_Digest

```

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

En este caso las subcategorias podrian ser el lenguaje, el puerto en el que se desplegara la aplicacion, el comando que se ejecutara al iniciar el contenedor, etc. Estos son elementos ya mencionados en la seccion de herramientas disponibles como pueden ser las acciones del comando docker init en Docker Desktop [8]. herramientas.

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

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

5.3.2. Plantillas Jinja2

Una vez que contamos con el modelo de la plantilla, podemos proceder a la creacion 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 Seccion de la Plantilla Dockerfile

²¹ La sintaxis de Jinja2 nos permite aplicar cierta lógica a la hora de generar los ficheros de configuracion, por ejemplo podemos emplear condicionales para que ciertas partes del fichero se generen o no en funcion de la configuracion que se haya dado.

```
{% include 'Common Templates/Metadata.jinja' %}
{% if Frontend %}
{% include 'Frontend/Frontend.jinja' %}
{% elif Backend %}
{% include 'Backend/Backend.jinja' %}
{% elif DataBase %}
{% include 'Database/Database.jinja' %}
{% elif DataBaseAdmin %}
{% include 'DataBaseAdmin/DataBaseAdmin.jinja' %}
{% elif OperatingSystem %}
{% include 'OperatingSystem/OperatingSystem.jinja' %}
{% endif %}
```

Figura 15: Captura de la primera subdivision de la plantilla dockerfile.

²¹Podemos 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

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 archivo aparte y reutilizarla en todas las plantillas que lo necesiten. Para este caso podemos generar una plantilla para los metadatos y labels de la imagen

```
1  {% if Metadata %}
2  {% if Header %}
3  #
4  #   This dockefile is generated by Automatic Dockerfile Generator v1.0
5  #
6  {% endif %}
7  # ----- Metadata -----#
8  {% if maintainer is defined %}
9  LABEL maintainer={{maintainer}}
10 {% endif %}
11 {% if version is defined %}
12 LABEL version={{version}}
13 {% endif %}
14 {% if description is defined %}
15 LABEL description={{description}}
16 {% endif %}
17 {% if ProjectName is defined %}
18 LABEL project={{ProjectName}}
19 {% endif %}
20 {% if license is defined %}
21 LABEL license={{license}}
22 {% endif %}
23 {% if build_date is defined %}
24 LABEL build-date={{build_date}}
25 {% endif %}
26 {% if author is defined %}
27 LABEL author={{author}}
28 {% endif %}
29 {% if email is defined %}
30 LABEL email={{email}}
31 {% endif %}
32 {% if repository is defined %}
33 LABEL repository={{repository}}
34 {% endif %}
35 {% endif %}
```

Figura 16: Captura de la primera subdivisión para dockerfile.

5.3.3. Mapping Model

El mapping model es un componente opcional que permite mapear features a otras features, esto es util cuando queremos renombrar features o hacerlas más comprensibles para las plantillas jinja. En este caso aplicamos dicha funcionalidad para paliar las restricciones del lenguaje uvl ²²

```
# 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 seccion del fichero Mapping Model plantilla Dockerfile.

²²En 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 nuestra intención a la hora de seguir desarrollar 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 por cada configuración y un producto esperado, si nuestra plantilla es capaz dada la configuración de entrada generar el mismo producto la prueba será considerada como exitosa.

```
1  {
2    "file": "Dockerfile fm.uvl",
3    "config": {
4      "Backend": true,
5      "Programming_Language": true,
6      "Python": true,
7      "Flask": true,
8      "Install_Python_Requirements": true,
9      "WSGI": true,
10     "Gunicorn": true,
11     "interface_Gunicorn": "0.0.0.0",
12     "workers_Gunicorn": "4",
13     "module_Gunicorn": "holamundo.py",
14     "callable_Gunicorn": "app",
15     "Gunicorn_Install": true
16   }
17 }
```

Figura 18: Configuración Ejemplo 1 Backend.


```

1  # Python Aplicacion Dockerfile
2  # Fask Aplicacion Dockerfile
3
4  # Avoid using latest tag in production
5  FROM python:latest
6
7  # Keeps Python from buffering stdout and stderr to avoid situations where
8  # the application crashes without emitting any logs due to buffering.
9  ENV PYTHONUNBUFFERED=1
10
11 # Create a non-privileged user that the app will run under.
12 # See https://docs.docker.com/go/dockerfile-user-best-practices/
13 ARG UID=10001
14 RUN adduser \
15     --disabled-password \
16     --gecos "" \
17     --home "/nonexistent" \
18     --shell "/sbin/nologin" \
19     --no-create-home \
20     --uid "${UID}" \
21     appuser
22 # Install the application dependencies
23 RUN --mount=type=cache,target=/root/.cache/pip \
24     python -m pip install --no-cache-dir -r requirements.txt
25
26 WORKDIR /app
27
28 COPY . /app
29 # Install Gunicorn This is not recommended for production, requirements.txt should be used instead
30
31 RUN pip install --no-cache-dir gunicorn
32
33 USER appuser
34
35 # Run the application using Gunicorn
36 CMD ["gunicorn", "-w", "4", "-b", "0.0.0.0:80", "holamundo.py:app"]
37
38 # Make port 5000 available to the world outside this container
39 EXPOSE 5000
40

```

Figura 19: Producto Esperado para la configuracion 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 generacion de los productos esperados. Estos test pueden encontrarse en el repositorio para el desarrollo de la plantilla dockerifle [28]

5.3.5. Resto de Plantillas

El proceso de creacion 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 configuracion que se pueden dar. Estas plantillas pueden encontrarse en el repositorio [30]

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. Vision 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

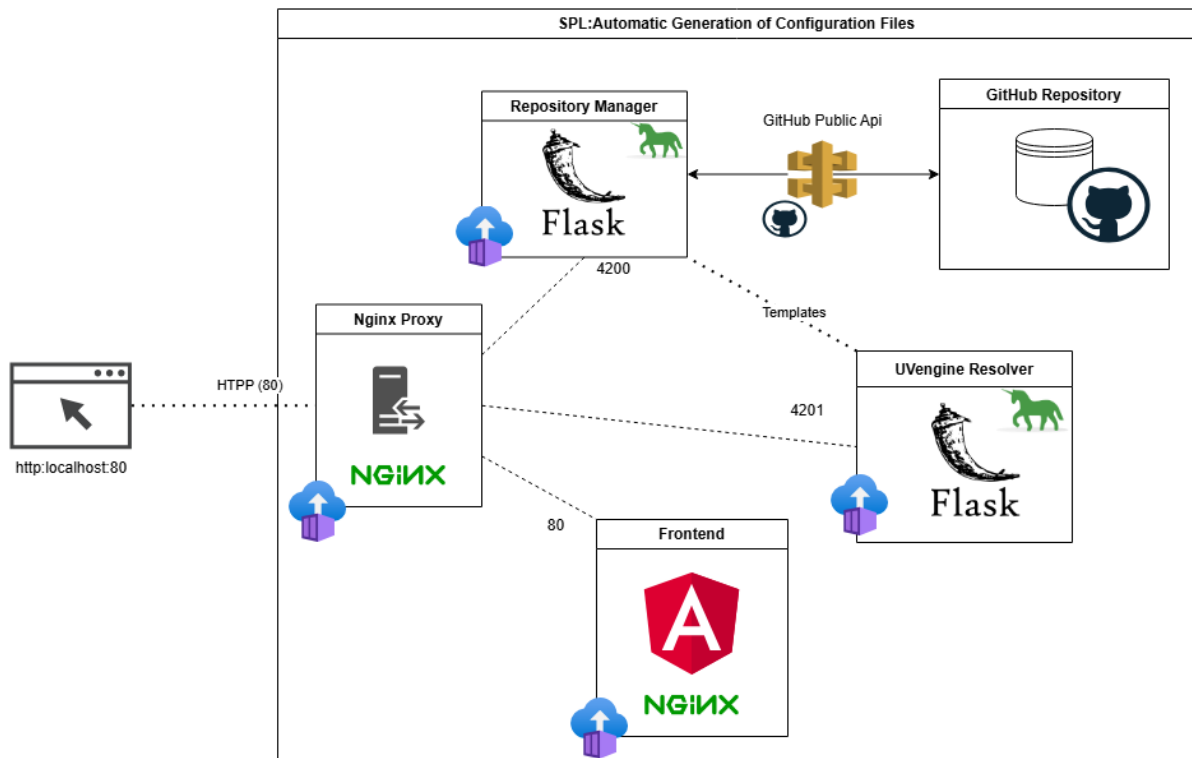


Figura 20: Diagrama infraestructura Contenedores

6.2. Descripción de servicios de backend

Para 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 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 [30] 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 ²³

²³En 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

La segunda de ellas es UVEngine-Resolver este servicio se encarga de resolver la variabilidad a partir de recibir como parametros un Json que contiene el nombre de la plantilla, la version a utilizar y la configuracion el resultado de la funcion sera un json que contiene el producto generado a partir de la configuracion dada.

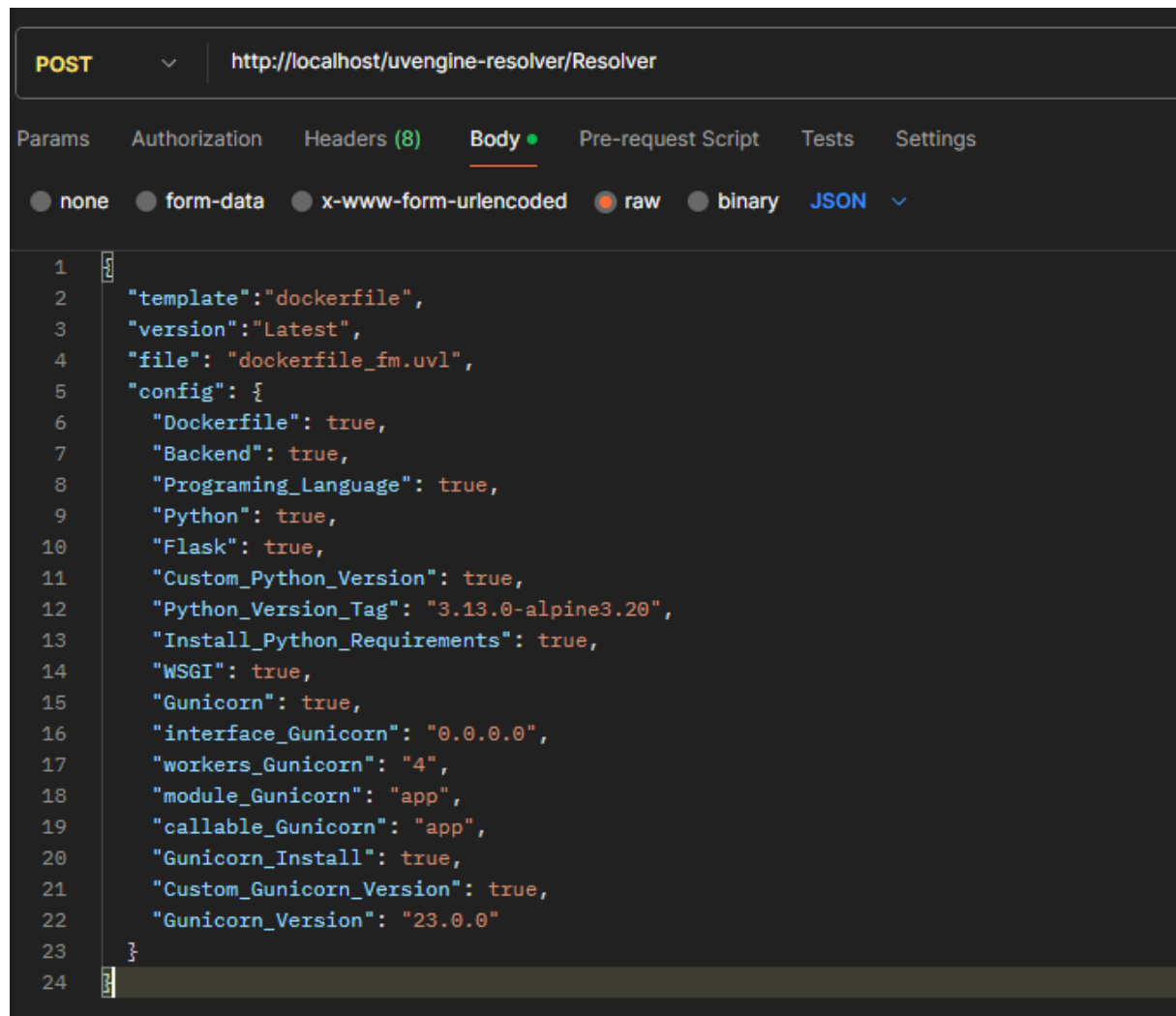


Figura 21: Ejemplo de una Petición POST a uvengine-resolver

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

6.3. Descripcion de Frontend

Como hemos comentado Anteriormente el Frontend es la parte de la aplicacion que se encarga de la interaccion con el usuario y la visualizacion de los archivos de configuracion generados. Se trata de una aplicacion web desarrollada en Angular que permite al usuario seleccionar una plantilla y una version y generar un producto a partir de una configuracion valida.

Esta Aplicacion queda alojada en un contenedor de Docker que se encarga de servir la aplicacion web en el puerto 80, este contenedor se comunica con el servicio Repository-Manager para obtener las plantillas y versiones disponibles a traves de peticiones del cliente. Para servir la aplicacion usamos el servidor web Nginx

Extensibilidad

En esta seccion se pretende hacer incapie en la extensibilidad de la aplicacion, como se ha mencionado anteriormente el proyecto tiene en mente que pueda adaptarse a las necesidades futuras.

Al almacenar las plantillas en un repositorio publico 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 aplicacion y que se puedan añadir nuevas funcionalidades sin necesidad de modificar el codigo de la aplicacion.

7.1. Añadir Plantillas

Existiendo varias formas en las que se pudiera haber diseñado la forma con la que añadir nuevas plantillas a la aplicacion 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 uvl, las plantillas Jinja2 y el archivo de Mapping Model, este ultimo archivo es opcional. Existen instrucciones detalladas de como añadir una plantilla al repositorio de plantillas ²⁴ o bien en el apendice Manual de Desarrollo

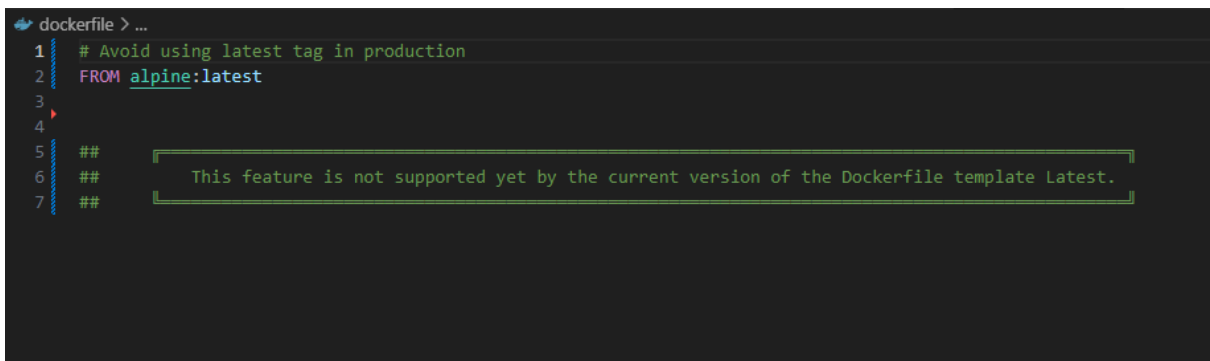
²⁴Repositorio de plantillas:<https://github.com/M4rdom/Templates>

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 ²⁵ 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



```
dockerfile > ...
1 # Avoid using latest tag in production
2 FROM alpine:latest
3
4
5 ##
6 ## This feature is not supported yet by the current version of the Dockerfile template Latest.
7 ##
```

Figura 22: Mensaje de Error en el Producto Final

²⁵Repositorio para el desarrollo de la Plantilla Dockerfile <https://github.com/M4rdom/Dockerfile-Template-Dev> [28]

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 multi-contenedor 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 parametros Un fichero Json que contiene el nombre de la plantilla, la version a utilizar y la configuracion
- Nginx Reverse Proxy: Servicio que se encarga de redirigir las peticiones de la aplicacion web al servidor de backend y fronte

La instruccion dockercompose up creara los contenedores en la configuracion ya escrita. La web sera accesible en la direccion 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 usuadas una vez generamos una configuración podemos simplemente descargar el fichero generado o copiarlo al portapapeles y quedaria 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 si. 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 adaptandola a las necesidades del proyecto.
- Se ha desarrollado una aplicación de Repository Manager que permite la comunicación con el repositorio de plantillas de GitHub [30] .
- 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 la plantilla auxiliares para ficheros de configuracion Nginx, dockerrignore, 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 ha aportado apendices para el uso más fundamental de la aplicacion.

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 mandatorio de la extension uvls para generar los ficheros de configuracion sea un punto en contra para la viabilidad del proyecto, El uso de la herramienta de configuracion puede no estar a la altura en cuanto a comodidad de uso de otras herramientas de generacion automatica de archivos. Las soluciones actualmente

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 que 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 en 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 [19] 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.

²⁶ 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 uv1 ni se podrá recargar el listado de plantillas y 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. uvengine-resolver.

²⁶ 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. Conclusion Personal

A nivel personal, el desarrollo de este trabajo me ha servido para introducirme de lleno en la tecnología de contenedores, como esta tecnología se aplica en servicios y que 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

Ademas de contar con una forma interesante de generar dockerfiles pueden existir dinamicas para aprovechar configuraciones validas para generar otros ficheros necesarios, como por ejemplo podemos ficheros de configuracion para el servidor de Nginx [35] directamente a partir de la configuracion 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 esta 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 configuracion como pueden ser en el caso de herramientas como Ansibl [44] Una vez añadidas plantillas al repositorio estan estaran disponibles de la misma forma en la que estan disponibles las plantillas existentes solo sera necesario recargar el listado de plantillas que se encuentra en el la barra de navegacion de la aplicacion web.

10.5. Continuación en el desarrollo de plantillas ya existentes

Las plantillas exitentes tienen un gran margen de mejora, En la situacion actual no soportan la gran mayoria de las opciones de configuracion que se pueden dar en un fichero Dockerfile. Además las opciones que se manejar son propensas a quedar obsoletas y se necesita de una continua revision de las plantillas en busca de problemas de diseño o seguridad.⁴ Estos son problemas que padecen otras herramientas que nos permiten generar el ficheros dockerfile de forma automatica.

10.6. Mejorar la experiencia del usuario

La generacion de la configuraciones estan ligadas al desarrollo de la extension UVLS, la extension permite al usuario generar una configuracion valida deplegando el arbol de caracteristicas pero para ello el usuario necesita contar con VSCode y la extension instalada para ser capaz de generar configuraciones validas. La propuesta para este escenario seria la integracion de dicha funcionalidad dentro de la plataforma web, la extension 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 [15].

Otras mejoras que supondrían una gran aportación sería desarrollar herramientas que permitan la generación de configuraciones de forma automática reconociendo las necesidades del proyecto, como hacen otras aplicaciones como Starter [46]

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] Docker. *Dockerfile Reference*. Accedido: 22 de octubre de 2024. 2024. URL: <https://docs.docker.com/reference/dockerfile/>.
- [10] Inc. Docker. *Docker Hub*. Accedido: 22 de octubre de 2024. 2024. URL: <https://hub.docker.com/>.

- [11] Inc. Docker. *Dockerfile Concepts*. Accessed: 2024-10-21. 2023. URL: <https://docs.docker.com/build/concepts/dockerfile/>.
- [12] Inc. Docker. *Intro Guide to Dockerfile Best Practices*. Accessed: 2024-10-21. 2023. URL: <https://www.docker.com/blog/intro-guide-to-dockerfile-best-practices/>.
- [13] Inc. Docker. *Use multi-stage builds*. Accedido: 22 de octubre de 2024. 2024. URL: <https://docs.docker.com/build/building/multi-stage/>.
- [14] Docker, Inc. *Docker: Accelerate How You Build, Share, and Run Modern Applications*. Accessed: 2024-10-21. 2024. URL: <https://www.docker.com/>.
- [15] Electron. *Electron: Desarrolla aplicaciones de escritorio con JavaScript, HTML y CSS*. Accedido: 22 de octubre de 2024. 2024. URL: <https://www.electronjs.org/es/>.
- [16] Python Software Foundation. *Python*. Accedido: 22 de octubre de 2024. 2024. URL: <https://www.python.org/>.
- [17] GitHub. *GitHub Personal Access Tokens*. Accedido: 22 de octubre de 2024. 2024. URL: <https://github.com/settings/tokens>.
- [18] Inc. GitHub. *GitHub*. Accedido: 22 de octubre de 2024. 2024. URL: <https://github.com/>.
- [19] Inc. GitHub. *GitHub REST API Documentation*. Accedido: 22 de octubre de 2024. 2024. URL: <https://docs.github.com/en/rest?apiVersion=2022-11-28>.
- [20] Google Cloud. *What are Containers?* Accessed: 2024-10-21. 2024. URL: <https://cloud.google.com/learn/what-are-containers?hl=es>.
- [21] Unicorn. *Unicorn*. Accedido: 22 de octubre de 2024. 2024. URL: <https://unicorn.org/>.
- [22] José Manuel Horcas. *UVEngine*. Accedido: 22 de octubre de 2024. 2024. URL: <https://github.com/jmhorcas/uvengine>.
- [23] José Manuel Horcas. *UVEngine 1.0.0*. Accedido: 22 de octubre de 2024. 2024. URL: <https://pypi.org/project/uvengine/1.0.0/>.

- [24] IBM. *Orquestación de contenedores*. Accedido: 21 de octubre de 2024. 2024. URL: <https://www.ibm.com/es-es/topics/container-orchestration>.
- [25] KeepCoding. *¿Qué es el archivo .dockerignore?* Accedido: 22 de octubre de 2024. 2024. URL: <https://keepcoding.io/blog/que-es-archivo-dockerignore/>.
- [26] 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/>.
- [27] 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>.
- [28] M4rdom. *Dockerfile-Template-Dev*. Accedido: 22 de octubre de 2024. 2024. URL: <https://github.com/M4rdom/Dockerfile-Template-Dev>.
- [29] M4rdom. *Frontend*. Accedido: 22 de octubre de 2024. 2024. URL: <https://github.com/M4rdom/Frontend>.
- [30] M4rdom. *Templates*. Accedido: 22 de octubre de 2024. 2024. URL: <https://github.com/M4rdom/Templates>.
- [31] MagickPen. *Plantilla para generar Dockerfiles*. Accedido: 21 de octubre de 2024. 2024. URL: <https://magickpen.com/templates/128/>.
- [32] 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>.
- [33] Microsoft. *Monaco Editor*. Accedido: 22 de octubre de 2024. 2024. URL: <https://microsoft.github.io/monaco-editor/>.
- [34] Microsoft. *Visual Studio Code*. Accedido: 22 de octubre de 2024. 2024. URL: <https://code.visualstudio.com/>.
- [35] Inc. Nginx. *Nginx*. Accedido: 22 de octubre de 2024. 2024. URL: <https://nginx.org/en/>.

- [36] Node.js. *Node.js*. Accedido: 22 de octubre de 2024. 2024. URL: <https://nodejs.org/en/>.
- [37] Inc. npm. *ngx-json-viewer*. Accedido: 22 de octubre de 2024. 2024. URL: <https://www.npmjs.com/package/ngx-json-viewer>.
- [38] Inc. npm. *npm*. Accedido: 22 de octubre de 2024. 2024. URL: <https://www.npmjs.com/>.
- [39] OpenAI. *ChatGPT*. Accedido: 21 de octubre de 2024. 2024. URL: <https://chatgpt.com/>.
- [40] Podman Project. *Podman: A Tool for Managing Containers and Pods*. Accessed: 2024-10-21. 2024. URL: <https://podman.io/>.
- [41] Pallets Projects. *Flask Documentation (3.0.x)*. Accedido: 22 de octubre de 2024. 2024. URL: <https://flask.palletsprojects.com/en/3.0.x/>.
- [42] Pallets Projects. *Jinja Documentation (3.1.x)*. Accedido: 22 de octubre de 2024. 2024. URL: <https://jinja.palletsprojects.com/en/3.1.x/>.
- [43] PyPI. *Python Package Index*. Accedido: 22 de octubre de 2024. 2024. URL: <https://pypi.org/>.
- [44] Inc. Red Hat. *Ansible Automation Platform*. Accedido: 22 de octubre de 2024. 2024. URL: <https://www.ansible.com/>.
- [45] Git SCM. *Git*. Accedido: 22 de octubre de 2024. 2024. URL: <https://git-scm.com/>.
- [46] Starter. *Starter: Herramienta de desarrollo*. Accedido: 22 de octubre de 2024. 2024. URL: <https://ww1.startwithdocker.com/?usid=15&utid=31190731855>.
- [47] UVL Playground. Accedido: 22 de octubre de 2024. 2024. URL: <https://uvl.uni-ulm.de/>.
- [48] 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 [27]²⁷ 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
```

²⁷<https://github.com/M4rdom/Automatic-generation-of-configuration-files-for-deploying-services-in-Docker>

A.4. Acceso a la aplicacion

Una vez que los servicios se hayan desplegado correctamente podremos acceder a la aplicacion web en la direccion localhost en el puerto 80

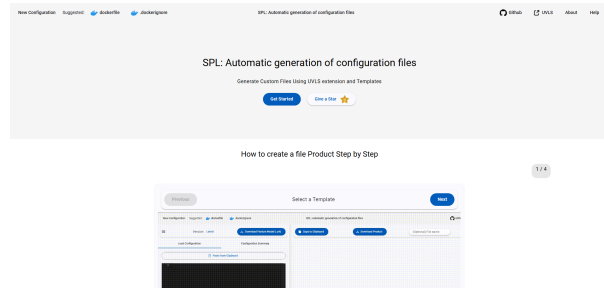


Figura 23: Pagina Principal

A.4.1. Github API

Debemos recordar que la aplicacion hace uso de la API publica de GitHub [19] para el servicio de Repository-manager, por lo que es posible que se alcance el limite de peticiones permitidas por la API. Para superar este limite se puede emplear un token de autenticacion de GitHub [17] para aumentar el numero de peticiones permitidas. Para un uso estadar no es necesario emplear un token de autenticacion.

A.5. Terminar el servicio

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

```
docker-compose down
```

Apéndice B

Manual de Uso

B.1. Inicio de la aplicacion

Para iniciar la aplicacion debemos acceder a la direccion localhost en el puerto 80

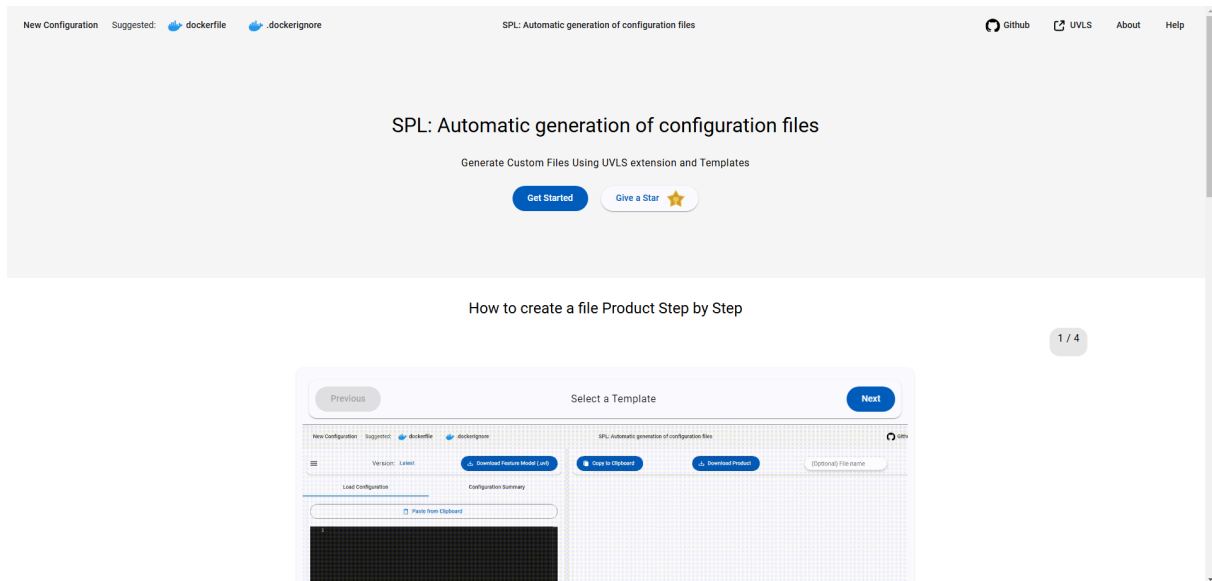


Figura 24: Pagina Principal

B.2. Barra de Navegación

La barra de navegacion de la aplicacion web contiene las siguientes opciones



Figura 25: Pagina Principal

- New Configuration: Permite al usuario seleccionar una plantilla y una version y generar un producto a partir de una configuracion valida.
- dockerfile: Permite al usuario crear un dockerfile a partir de una configuracion valida.

- `.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

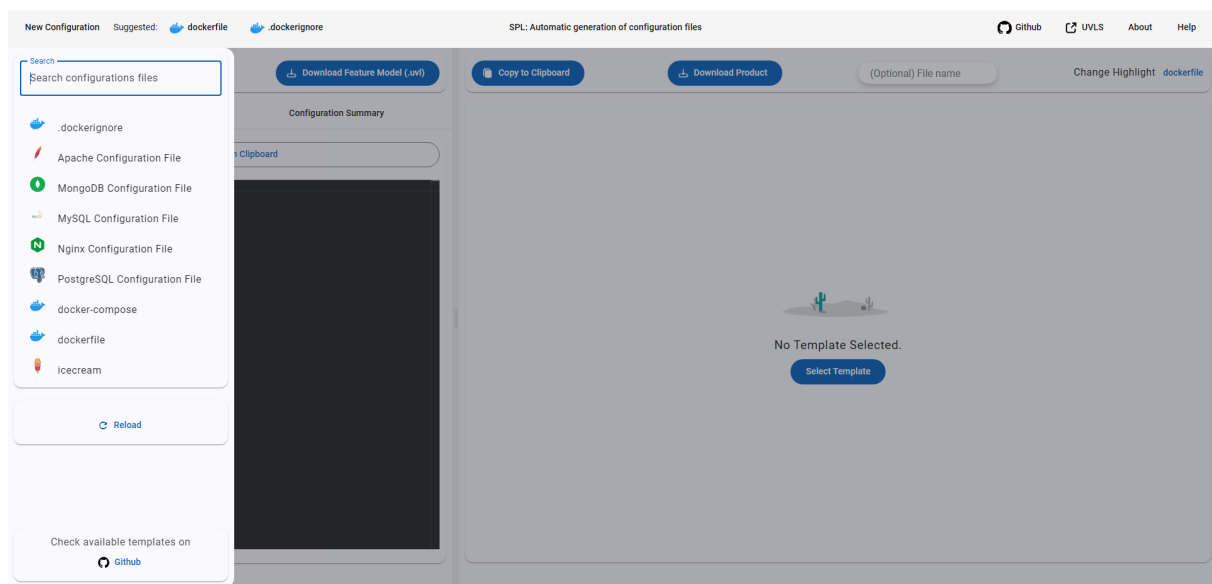


Figura 26: Barra Desplegable Selector de Plantillas

B.3.1. Seleccionar version de la platilla

Para seleccionar al version de la plantilla abrimos el desplegable en Version por defecto tiene el valor de Latest

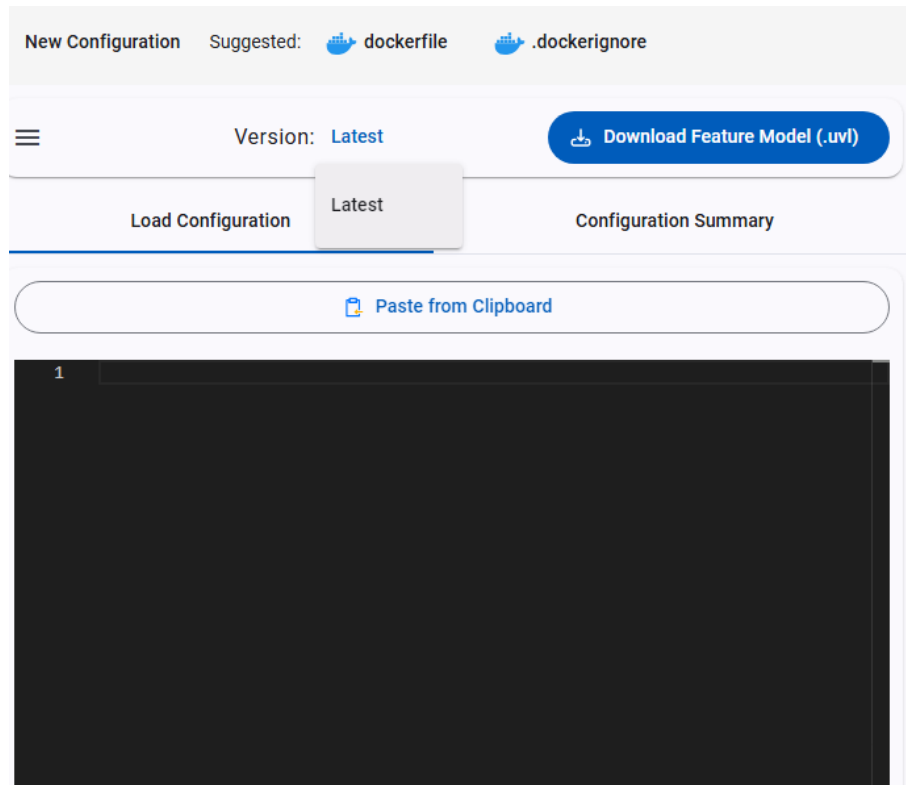


Figura 27: Captura Versiones Disponibles

B.3.2. Refrescar Plantillas

En la pagina de arriba tendremos la opcion de actualizar la lista de plantillas disponibles haciendo click en el boton Reload

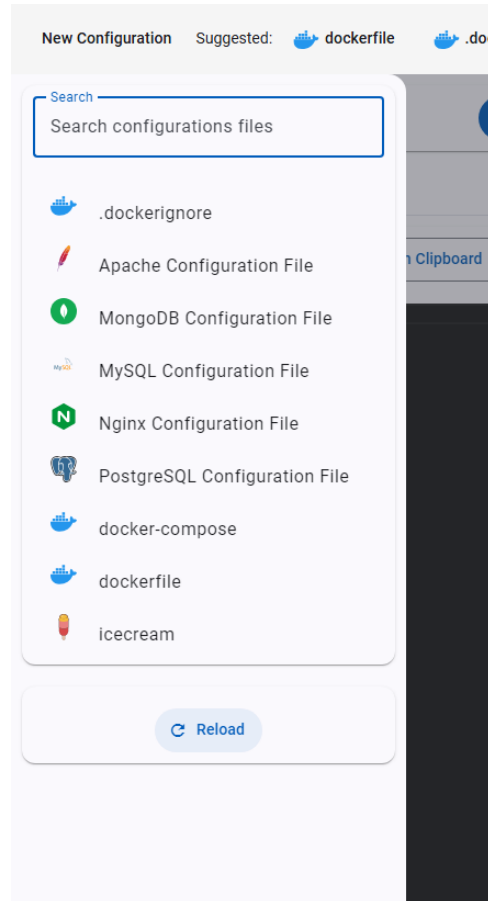


Figura 28: Captura Versiones Disponibles

B.3.3. Cargar configuracion

Para cargar una configuracion debemos hacer click en el boton Load Configuration y copiar en el editor un fichero de configuracion valido Opcinalmente podemos pegar la configuracion desde el portapapeles

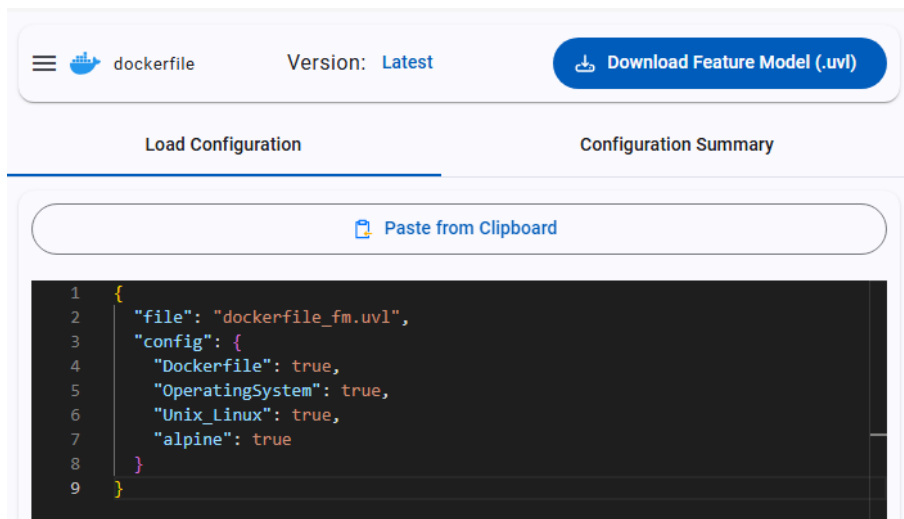


Figura 29: Editor para cargar configuracion

B.3.4. Descargar Fichero Generado

Una vez que hemos cargado la configuracion y seleccionado una plantilla y una version podemos descargar el fichero generado haciendo click en el boton Download File y opcionalmente añadiendo un nombre al fichero de salida. O bien simplemente copiamos al portapapeles

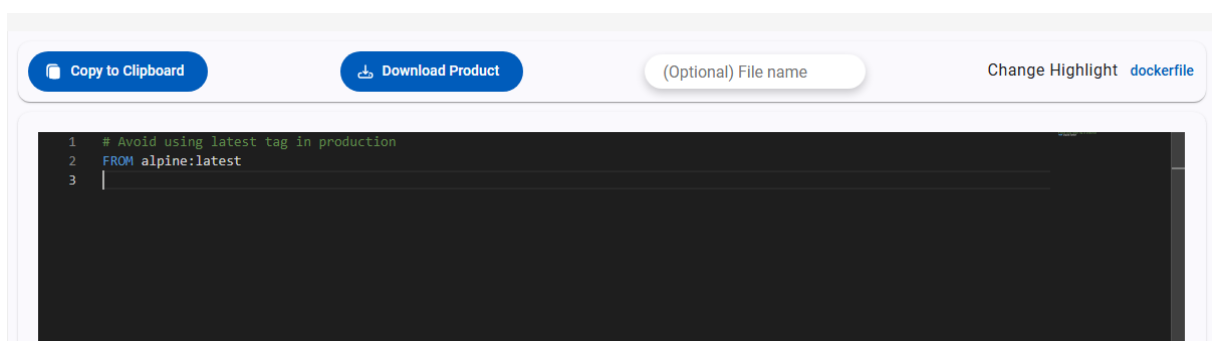


Figura 30: Editor para guardar Configuracion

B.3.5. About Page

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

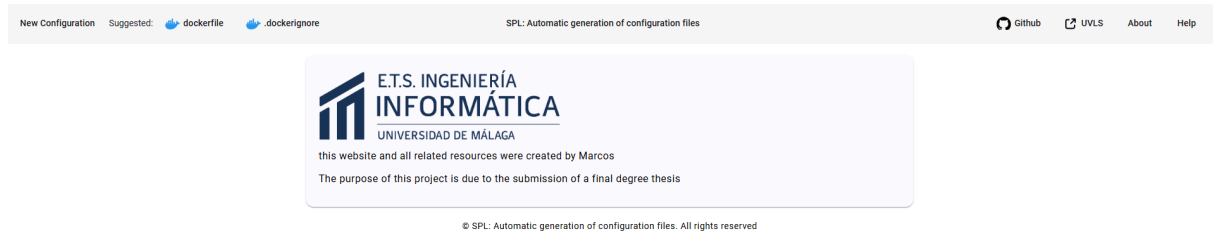


Figura 31: Pagina Principal

B.3.6. Help Page

Help Page contine informacion sobre la terminologia del proyecto

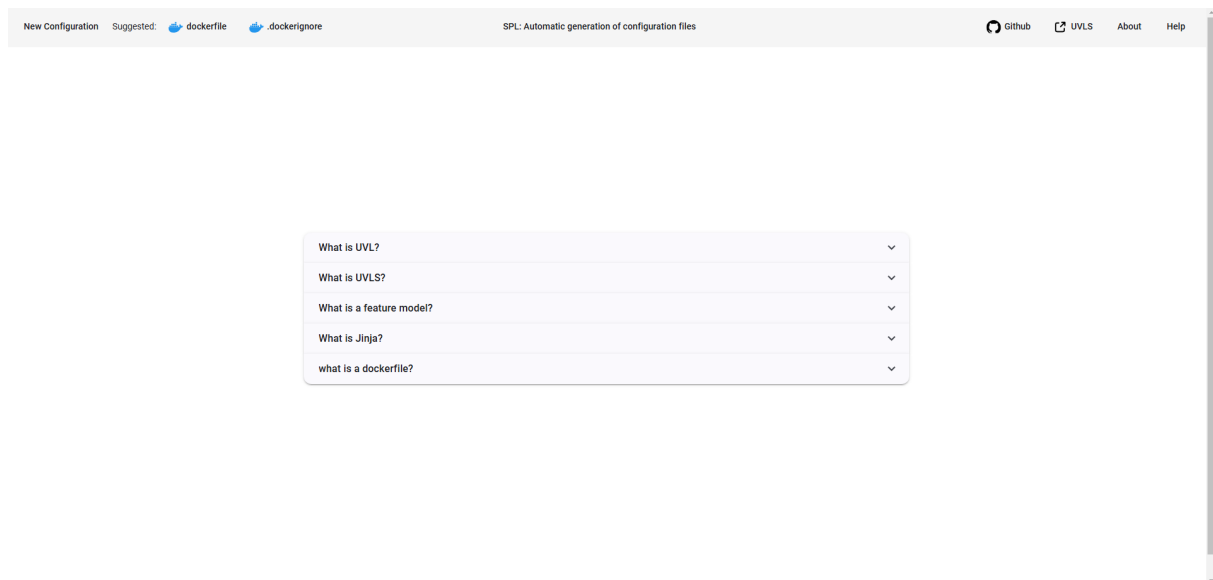


Figura 32: Captura Pagina de ayuda

Apéndice C

Manual Desarrollo de Plantillas

C.1. Introducción

Este manual describe los pasos necesarios tanto para añadir nuevas plantillas al repositorio de plantillas [30] como para modificar las plantillas existentes. una vez añadidas las plantillas al repositorio sera necesario actualizar la lista de plantillas dentro de la aplicacion vease el Mnaual 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 Exstructura 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 version
 - 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 Feature 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 aplicacion 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 aplicacion web

Es recomendable generar estas plantillas a partir de un repositorio de desarrollo [28] para asegurar que las plantillas generan productos validos realizando pruebas. Una vez que tengamos una plantilla lista, es solo cuestion de subirla al repositorio respetando la extrcutura original modificando unicamente la unica plantilla que se desea modificar. Estas Instrucciones pueden verse con más detalle en el repositorio de plantillas [30]



UNIVERSIDAD
DE MÁLAGA

| **uma.es**

E.T.S. DE INGENIERÍA INFORMÁTICA

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