



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

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Keywords: A, B, C

Resumen

Por revisar

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

Índice

1. Introduction	11
1.1. Motivación	11
1.2. Objetivo	12
1.3. Resultados Esperados	12
1.4. Estructura del documento	12
2. Estado del Arte	13
2.1. Contenedores	13
2.1.1. Porque se usan los contenedores	13
2.2. Imagenes	13
2.3. Docker	13
2.3.1. Dockerfile	14
2.3.2. Buenas Prácticas Docker Images	14
2.4. Problematica	14
2.5. Soluciones Actualmente Disponibles	14
2.5.1. Docker init	15
2.5.2. Docker Extension for Visual Studio Code	15
2.5.3. Inteligencia Artificial	15
2.5.4. Starter	16
2.6. Conclusiones	16
2.7. Que se aporta en este contexto:	17
2.7.1. Software Product Line SPL	17
3. Tecnologías Empleadas	19
3.1. Universal Variability Language UVL	19
3.1.1. Feature Models	19
3.2. Docker	19
3.3. Node.js	20

3.4.	Angular	20
3.4.1.	Angular Material	20
3.4.2.	Monaco Editor	21
3.4.3.	ngx-json-viewer	21
3.5.	Nginx	21
3.6.	Python	21
3.6.1.	Flask	22
3.6.2.	guinicorn	22
3.6.3.	Pypi	22
3.7.	Jinja	22
3.7.1.	Jinja2	22
3.8.	Git	23
3.9.	GitHub	23
3.9.1.	GitHub Public Api	23
3.10.	Visual Studio Code	24
3.10.1.	UVLS extension	24
4.	Metodología del trabajo. (Fases) Implementación:	25
4.1.	Introduction	25
4.2.	Metodología del Desarrollo	25
4.2.1.	Justificación	26
4.2.2.	Procedimiento	26
4.2.3.	Limitaciones	26
4.3.	Fases de Estudio	26
4.4.	Estudio del Estado del Arte :	26
4.5.	Investigación dockefiles	26
4.6.	Desarrollo de Modelos de plantillas	27
4.7.	Fase desarrollo Scrum	27
4.7.1.	Requisitos de la aplicación	27
4.7.2.	Desarrollo del la aplicación de uvengine	27
4.7.3.	Desarrollo de la web	28

4.7.4.	Desarrollo de los servicios de backend	28
4.7.5.	Elaboración de la guía de uso	28
4.7.6.	Elaboración de un manual de instalación	28
4.8.	Elaboración de la memoria	28
5.	Modelado de las plantillas	29
5.1.	Introducción	29
5.2.	Componentes de las plantillas	29
5.2.1.	Featuremodel(.uvl)	29
5.2.2.	Plantillas Jinja	30
5.2.3.	Mapping Model	30
5.3.	Plantilla dockerfile	30
5.3.1.	Plantillas Jinja2	31
5.3.2.	Test sobre la Plantilla dockerfile	32
6.	Arquitectura y Descripcion del sistema	37
6.1.	Vision general de la arquitectura	37
6.2.	Descripcion de servicios de backend	37
6.2.1.	Repository Manager	37
6.2.2.	UVEngine Resolver	38
6.3.	Descripcion de Frontend	38
7.	Extensibilidad	39
7.1.	Añadir Plantillas	39
7.1.1.	Ampliar la Plantilla Dockerfile	39
8.	Despliegue de la aplicación	41
8.1.	Despliegue local	41
8.2.	Despliegue en la nube	43
9.	Resultados	45
9.1.	Resultados Obtenidos	45
9.2.	Viabilidad del proyecto	45

9.3. Escalabilidad	46
10. Conclusiones y líneas Futuras	47
10.1. Conclusiones del Desarrollo del proyecto	47
10.2. Conclusion Personal	47
10.3. Aplicaciones Prácticas	48
10.4. Desarrollo de nuevas plantillas	48
10.5. Continuación en el desarrollo de plantillas ya existentes	48
10.6. Mejorar la experiencia del usuario	48
Apéndice A. Manual de	
Instalación	51
A.1. Requisitos Previos	51
A.2. Descargar el codigo fuente	51
A.3. Configuración del archivo Dockercompose	51
A.3.1. Nginx reverse proxy	51
A.3.2. frontend	51
A.3.3. Repository Manager	51
A.3.4. UV Engine resolver	51
A.4. Construcción de los contenedores	51
A.4.1. Github API	51
A.5. Terminar el servicio	51
Apéndice B. Manual de Uso	53
B.1. Inicio de la aplicación	53
B.2. Barra de Navegación	53
B.2.1. About Page	53
B.2.2. Help Page	53
B.3. Seleccionar plantilla	53
B.3.1. Seleccionar versión de la plantilla	53
B.3.2. Cargar configuración	53
B.3.3. Descargar Fichero Generado	53

B.3.4. Copiar Fichero Generado	53
--	----

1

Introduction

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 los archivos Dockerfile y .dockerignore, que definen cómo se construye una imagen de un contenedor Docker.

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

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

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

1.2. Objetivo

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

1.3. Resultados Esperados

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

1.4. Estructura del documento

Por definir

Estado del Arte

En este apartado se presentan una introducción a los conceptos clave necesarios para comprender la situación actual y contextualizar la problemática, así como las soluciones existentes.

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.

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. Es una solución de fácil aplicación, alto rendimiento y extremadamente portable.

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. CITA

2.3. Docker

Docker CITA 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.

2.3.1. Dockerfile

El archivo Dockerfile 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. Buenas Prácticas Docker Images

Para la creación de los ficheros dockerfile es recomendable seguir una serie de buenas prácticas para garantizar la eficiencia y la seguridad en el despliegue de servicios en Docker. Estas prácticas se recogen en la documentación oficial de Docker CITA

2.4. Problemática

La dificultad de crear imágenes reside en la naturaleza de la aplicación que queremos llevar a un contenedor, existen una gran cantidad de escenarios donde si bien la tarea es sencilla, esta puede esconder una complejidad implícita de la que el usuario no es consciente.

Estos apartados repercuten en el tiempo de compilación, tamaño de la imagen, mantenibilidad, seguridad y repetibilidad en las imágenes que se crean.

La calidad de las imágenes pueden marcar una diferencia que puede llegar a ser crítica para posteriores etapas en el desarrollo de aplicaciones como son la Orquestación de contenedores CITA.

2.5. Soluciones Actualmente Disponibles

En la actualidad hay disponibles varias soluciones para disminuir la complejidad implícita que reside en la configuración de dichos ficheros dockerfile

2.5.1. Docker init

Docker init es el comando de Docker Desktop CITA 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.

```
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 CITA permite entre sus funcionalidades la generacion de los ficheros dockerfile y dockerignore. 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.

2.5.3. Inteligencia Artificial

En la actualidad existen soluciones que se apoyan en la inteligencia Artificial para la generacion de archivos de configuracion, y para la creacion de los fichero 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 promp de entrada. como <https://magickpen.com/templates/128/> CITA o alternativas con proposito más general como chatgpt CITA

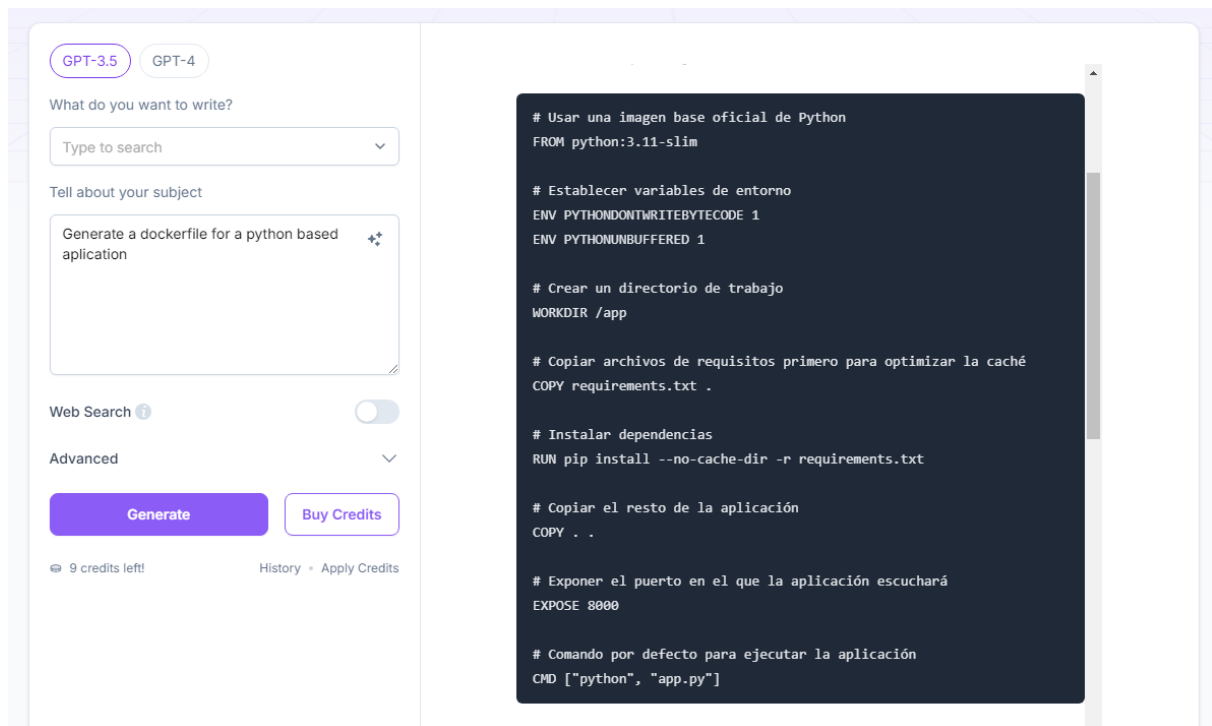


Figura 2: Captura de Magickpen.

2.5.4. Starter

Starter es una herramienta open source CITA <https://www.startwithdocker.com/> capaces de reconocer el lenguaje de programacion del proyecto y generar un archivo de configuracion. La herramienta es capaz de forma autonoma reconocer el contenido de un proyecto y generar los ficheros dockerfile y dockercompose

2.6. Conclusiones

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 generacion de los ficheros dockerfile y dockerignore, estas plantillas si bien son correctas son por naturaleza limitadas y no permiten la personalizacion más final de las opciones de configuracion. 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

El proyecto propone una solución en línea con la implementación de plantillas pero ... 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) 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. CITA

IMAGEN FEATURE MODEL

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

CITA La Universal Variability Language (UVL) es un lenguaje de modelado diseñado específicamente para representar y manejar la variabilidad en sistemas software. 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 (Software Product Line Engineering o SPLE), 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.

3.1.1. Feature Models

CITA Un **Feature Model** es una representación de las características y variabilidad de un producto o sistema. En el caso de una aplicación de mensajería móvil, el modelo podría ser el siguiente:

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 utilización de Docker Desktop es complemento con la consola de comandos. Esta herramienta ayuda al proyecto en el momento en que lo hacer portable y fácil de desplegar en cualquier entorno.

3.3. Node.js

CITA para el desarrollo de la aplicación web se ha optado por la utilización de Node.js como entorno de ejecución de JavaScript para el desarrollo 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), Node.js ofrece acceso a una vasta colección de paquetes y módulos que pueden ser fácilmente integrados en el proyecto, acelerando el desarrollo y reduciendo la necesidad de escribir código desde cero. Para el proyecto se emplea la versión v22.10.0 y la versión v10.9.0 del gestor de paquetes npm

3.4. Angular

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

3.4.1. Angular Material

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

aplicacion web.

3.4.2. Monaco Editor

CITA 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.

3.4.3. ngx-json-viewer

CITA 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

CITA 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

CITA 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

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

CITA 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

CITA 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

CITA 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

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

3.7.1. Jinja2

CITA 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

CITA Para el control de versiones se ha optado por la utilizacion de Git como sistema de control de versiones.

3.9. GitHub

CITA 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

CITA 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. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

3.10. Visual Studio Code

CITA 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

CITA UVLS es una extensión de Visual Studio Code que proporciona soporte para la Universal Variability Language (UVL) en el editor de código. La extensión UVLS proporciona resaltado de sintaxis, autocompletado, verificación de errores y otras funcionalidades que facilitan la edición de archivos de configuración en UVL. Además la clave más importante es que es la única herramienta existente que permite crear una configuración correcta partiendo desde un Feature Model

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

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

4

Metodología del trabajo. (Fases) Implementación:

4.1. Introduction

El enfoque se ha dado para abordar el desarrollo de este trabajo ha consistido en combinar un trabajo previo de analisis e investigacion con un desarrollo iterativo y agil. La investigacion y analisis tenia el objetivo de conocer en profundidad las tecnologias y herramientas empleadas para porstesiriomete 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 agil que permita la adaptacion a los cambios y la evolucion del proyecto.

4.2. Metodología del Desarrollo

Al encontrarnos ante un trabajo individual toda la responsabilidad y capacidad del desarrollo recae sobre una persona. Para el desarrollo de código optaremos por una metodología Scrum simplificada. De forma iterativa se ejecutará un Sprint con unos objetivos de desarrollo y planificación muy limitados. Con el objetivo de dar cabida a la complejidad del proyecto, con animo de encajar nuevas ideas y revisiones que pudieran sucederse. Tras cada Sprint se evaluará la progresión y dirección del proyecto con el tutor.

4.2.1. Justificación

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. De entrada aportar una solución técnica óptima a un problema es extremadamente improbable, por lo que se opta por una metodología de desarrollo ágil que permita la adaptación a los cambios y la evolución del proyecto.

4.2.2. Procedimiento

Tras cada Sprint se evaluará la progresión y dirección del proyecto con el tutor.

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 configuración.

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.

4.5. Investigación dockefiles

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

yecto software. Es necesario comprender el porque se justifica la existencia de esta tecnologia y como se aplica a proyectos reales como los recogidos a continuacion CITA

4.6. Desarrollo de Modelos 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 pondran en practica todo el conocimiento adquirido en los pasos anteriores que tendran como resultado el modelado de unas plantillas. Hemos de tener en cuenta las buenas practicas y las recomendaciones de la comunidad para la creacion de los ficheros dockerfile CITA. Estas practicas tienes que estar presentes en el diseño de las plantillas desde el inicio, tambien hemos de generar test para mantener la consistencia de las implementaciones. Este paso queda cubierto con más detalle en el Capitulo 5

4.7. Fase desarrollo Scrum

4.7.1. Requisitos de la aplicación

Aquí se definen y documentan los requisitos funcionales y no funcionales de la aplicación. Se recopilan y analizan las necesidades de los posibles usos, los objetivos del proyecto y las restricciones técnicas para establecer una planificacon del desarrollo. Los requisitos han cambiado conforme el proyecto avanzaba en sus etapas de desarrollo sin animo de ser exhaustivos las funcionalidades mínimas que se pretenden alcanzar son en cuanto a la generacion de productos

La aplicacion web debe permitir seleccionar la plantilla y la version con la que se desea trabajar Las plantillas pueden recargarse en del repositorio La aplicacion web debe generar un el producto en el momento en el que se aplique una configuracion valida

4.7.2. Desarrollo del la aplicación de uvengine

En esta fase se desarrolla la aplicación de uvengine, que permite la generación automática de archivos de configuración

4.7.3. Desarrollo de la web

En esta fase se desarrolla la aplicación web que permite la interacción con el usuario y la visualización de los archivos de configuración generados.

4.7.4. 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.

4.7.5. 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.

4.7.6. Elaboración de un manual de instalación

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

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.

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 apliacion 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

Las plantillas se dividen en varios componentes

5.2.1. Featuremodel(.uvl)

El archivo uvl es el archivo principal de la plantilla, contiene la descripcion de las caracteristicas y las restricciones de la plantilla. en este archivo dicta principalmente como se generera una configuracion valida que sea capaz de cumplir con las restricciones textuales IMAGEN

CITA

5.2.2. Plantillas Jinja

Las plantillas Jinja son los archivos de plantilla que se utilizan para generar los productos a partir de una configuración válida o no del modelo. CITA La idea es exponer estas plantillas ante una configuración ya validada con herramientas como UVLS CITA, Estas plantillas constituyen el núcleo del funcionamiento, en ella se declaran de forma explícita las opciones de configuración y las variables que se pueden modificar. Las plantillas tienen que tener la estructura necesaria como para Usando la Sintaxis de Jinja crear un producto válido para una configuración existente

5.2.3. Mapping Model

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

5.3. Plantilla dockerfile

Como hemos visto un fichero Dockerfile es un archivo de texto que contiene una serie de instrucciones que Docker CITA utilizará para construir una imagen de contenedor. la plantilla para dockerfile cuenta con el modelo más grande del proyecto, La idea para crear estos modelos 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 CITA donde divide sus imágenes por categorías dependiendo de su función. En este contexto podemos construir una primera subdivisión que represente el resultado final esperado, es decir la primera feature será decidir si nuestra imagen tiene el objetivo de desplegar un contenedor que albergue un Frontend, Backend, Base de datos, etc.

```

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 3: Captura de la primera subdivision para dockerfile.

Cada una de estas categorias contendra a su vez una serie de subcategorias que representen 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 CITA 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

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 Docker Init CITA, Nuestro modelo permite recoger estas caracteristicas además de otras como pueden ser el servidor web 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 alla de lo que se recoge en esta version del fichero dockerfile pero estas opciones ya expanden enormemente la capacidad de configuracion que tienen otras herremientas.

5.3.1. 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. 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. Dividir las plantillas no solo es util porque las hace más manejables y faciles de mantener, sino que tambien permite la reutilizacion de

```

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 4: Captura de la primera subdivision para dockerfile.

las mismas, por ejemplo si tenemos una plantilla que se repite en varias plantillas, podemos extraerla a un fichero aparte y reutilizarla en todas las plantillas que lo necesiten. Para este caso podemos generar una plantilla para los metadatos y labels de la imagen

5.3.2. Test sobre la Plantilla dockerfile

El diseño y construccion de las plantillas es un proceso incremental, debe comprobarse con regularidad si nuestra intencion a la hora de seguir desarrollar una plantilla es mantener una coherencia con lo añadido hasta el momento. Con esta idea en mente la solucion propuesta consiste en la creacion de pruebas que validen la correcta construccion de productos segun la plantilla en la que estamos trabajado. 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 configuracion y un producto esperado, si nuestra plantilla es capaz dada lo configuracion de entrada generar el mismo producto la prueba sera considerada como exitosa.


```

1  {% if Metadata %}
2  {% if Header %}
3  #
4  # This dockfile 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 5: Captura de la primera subdivision para dockerfile.

```

{% 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 6: Captura de la primera subdivision para dockerfile.

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 CITA para la generación de los productos esperados. para cada una de las primeras divisiones de las plantillas se generan ficheros de test para clasificar las configuraciones que se prueban y los productos esperados.

```

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 7: Captura de la primera subdivision para dockerfile.

```

1 {
2   "file": "Dockerfile_fm.uvl",
3   "config": {
4     "Backend": true,
5     "Programing_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 8: Conguracion 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 9: Producto Esperado.

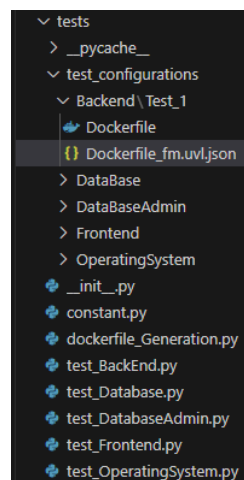


Figura 10: Captura de la primera subdivision para dockerfile.

6

Arquitectura y Descripcion del sistema

Esta seccion describe la arquitectura y el modelado del sistema, incluyendo los componentes, la estructura y el diseño de la aplicación resultante.

6.1. Vision general de la arquitectura

La arquitectura de la aplicación se divide en dos partes principales: el backend y el frontend. El backend es responsable de la generación de los archivos de configuración a partir de las plantillas y de la comunicación con los repositorios de plantillas. El frontend es responsable de la interacción con el usuario y la visualización de los archivos de configuración generados.

6.2. Descripcion de servicios de backend

Para los servicios de backend estos se han dividido en varios servicios que se encargan de tareas específicas. La primera de ellos es Repository Manager CITA este servicio se encarga de servir traducir las peticiones que puede realizar el front end a la API de GitHub. Estas peticiones son descargar el archivo uvl de la plantilla y version seleccionada, descargar el listado de plantillas disponibles y descargar el listado de versiones disponibles para cada plantilla

6.2.1. Repository Manager

Esta aplicaicon esta desarrollada en Python y emplea el framework Flask para la creacion de la API, esta API se encarga de servir los archivos uvl de las plantillas y las versiones dispo-

nibles. para hacer funcionar esta API se emplea la api publica de github que permite acceder al contenido de repositorios. En este caso el repositorio al que queremos acceder es el repositorio Templates CITA que contiene las plantillas disponibles.

6.2.2. UVEngine Resolver

La segunda de ellas es UV Engine Resolver CITA este servicio 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 el resultado de la funcion sera un producto

6.3. Descripcion de Frontend

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, se permite a cualquier usuario modificar, añadir o eliminar plantillas de forma sencilla. Esot permite que el desarrollo de plantillas se maneje de forma independiente al resto de la aplicacion y que se puedan añadir nuevas plantillas 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 solucion 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 especifica que contenga los archivos necesarios para la plantilla como son el archivo uvl, las plantillas Jinja2 y el archivo de Mapping Model, este ultimo archivo es opcional. Existen instucciones detalladas de como añadir una plantilla al respositorio de plantillas en CITA

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 todavia existe mucho margen de mejora en cuanto a la personalizacion de las opciones de configuracion. La plantilla no es a dia de hoy capaz de cubrir todas las opciones de configuracion 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 CITA 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.

8

Despliegue de la aplicación

8.1. Despliegue local

Para desplegar el proyecto y todas sus capas se hace uso de la herramienta Docker CITA, 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 CITA, 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

8.2. Despliegue en la nube

Por definir

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, la escalabilidad y las posibles mejoras futuras.

9.1. Resultados Obtenidos

La implementacion de la aplicacion aporta una forma agil de crear productos software a partir de ellas. Las plantillas para generar ficheros como dockerfiles recogen una un interesante número de posibilidades entre las que se encuentran las opciones más usuadas una vez generamos una configuracion podemos simplemente descargar el fichero generado o copiarlo al portapapeles y quedaria listo para su uso. El empleo de la extension uvls para generar una configuracion puede ser muy eficiente a la hora de generar productos con opciones de configuracion relacionadas entre si. Ademas tenemos la posibilidad de editar el producto final antes. Las plantillas existentes aportan unas funcionalidades interesantes que pueden explotarse a la hora de genera productos como ficheros de Configuracion. El repositorio es facilmente ampliable para abarcar nuevas plantillas y desarrollar nuevas funcionalidades.

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, pese a que la extension es un proyecto de codigo abierto y seria posible su integracion en la aplicacion. Las soluciones actualmente disponibles son más cómodas al detectar las features CITA del proyecto de forma autonoma sin necesidad de intervencion 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 CITA para el servicio de Repository manager CITA que limita el número de peticiones que se pueden hacer en un periodo de tiempo. 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. Actualmente el número de peticiones que puede realizarse a la API son 80 peticiones por minuto CITA 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 proyecto 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 proyecto 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 CITA 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 no solo para la generacion de ficheros dockerfile y algunos de los archivos de configuracion sino que se puede extender a otros tipos de ficheros, como por ejemplo ficheros de configuracion de Kubernetes CITA, ficheros de configuracion de Jenkins CITA, etc. 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.

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 CITA 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 codigo abierto y su integracion en la aplicacion web podria ser un proyecto en si

mismo.

En el desarrollo del trabajo se ha explorado esta posibilidad ejecutando dicha extension en la version web del editor VS code pero el resultado en cuanto a rendimiento hace que esta opcion no sea viable.

La propia aplicacion web puede ser mejorada en cuanto a la experiencia del usuario añadiendo más informacion al usuario para colocar más contexto a la hora de seleccionar una plantilla y las posibilidades de combianar configuraciones

Apéndice A

Manual de Instalación

A.1. Requisitos Previos

A.2. Descargar el código fuente

A.3. Configuración del archivo Dockercompose

A.3.1. Nginx reverse proxy

A.3.2. frontend

A.3.3. Repository Manager

A.3.4. UV Engine resolver

A.4. Construcción de los contenedores

A.4.1. Github API

A.5. Terminar el servicio

Apéndice B

Manual de Uso

B.1. Inicio de la aplicacion

B.2. Barra de Navegación

B.2.1. About Page

B.2.2. Help Page

B.3. Seleccionar plantilla

B.3.1. Seleccionar version de la platilla

B.3.2. Cargar configuracion

B.3.3. Descargar Fichero Generado

B.3.4. Copiar Fichero Generado



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