UNIVERSIDAD DE SANTIAGO DE COMPOSTELA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA

Ciclo completo de CI/CD con Dagger y Kubernetes

Autor/a:
Daniel Vieites Torres

Tutores:
Juan Carlos Pichel Campos
Francisco Maseda Muiño

Grado en Ingeniería Informática

2025

Trabajo de Fin de Grado presentado en la Escuela Técnica Superior de Ingeniería de la Universidad de Santiago de Compostela para la obtención do Grado en Ingeniería Informática

Resumen

Este trabajo aborda la gestión completa de un ciclo CI/CD (*Continuous Integration/Continuous Delivery*) con Dagger. Se desarrolla una aplicación de prueba, que consta de un frontend y un backend, junto con la infraestructura como código, y se comparan dos *pipelines*: uno implementado con Dagger y otro sin él. Los resultados demuestran que Dagger mejora el flujo de trabajo debido a la gestión que realiza de la caché, ejecutando todo el ciclo de CI/CD un 80 % más rápido que el *pipeline* sin Dagger. Como resultado, este trabajo propone un conjunto de módulos de Dagger que ofrece un enfoque práctico sobre cómo utilizar Dagger para acelerar el desarrollo despliegue de cualquier aplicación, minimizando el tiempo de espera y pasos manuales.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introdución

1.1. Objetivos generales

Objetivos principales

En este trabajo se pretende demostrar y evaluar la viabilidad, eficiencia y flexibilidad de Dagger[1] como motor de CI/CD (*Continuous Integration/Continuous Delivery*)[2, 4], con el fin de estandarizar y modernizar los ciclos de vida del desarrollo de software.

No solo se va a utilizar Dagger como complemento del ciclo de desarrollo de una aplicación, sino que se va a comparar con la no utilización de este. Se evaluarán las ventajas que tiene su uso frente a métodos convencionales, entre las que destacarán la gestión de caché y el uso de contenedores.

Se van a proporcionar ejemplos de módulos creados con Dagger, los cuales estarán especialmente diseñados para cumplir los ciclos tanto de CI como de CD de una aplicación *dummy*. De esta manera se podrá comprobar que este mismo proceso se puede llevar a cabo para cualquier aplicación, y de una manera sencilla.

Objetivos secundarios

Para lograr los objetivos principales es necesario llevar a cabo varios pasos:

- Creación de un monorepo[5] en GitHub.

  Todo el código principal se almacenará en un mismo repositorio. De esta manera se evitarán complicaciones debido a la gestión de dependencias de cada una de las partes de la aplicación. Permitirá controlar todo el código fuente de una manera más sencilla al tratarse de un proyecto relativamente pequeño.

- Diseño y creación de una aplicación *dummy*.

  Es necesario una aplicación sobre la que realizar los ciclos de CI/CD. Esta consistirá en una página web (frontend) de gestión de un zoo, la cual rea-

lizará peticiones a una API (backend) que estará conectada a una base de datos.

- Implementación de un *pipeline* CI/CD.

  Se creará un módulo de Dagger para cada uno de los procesos (CI y CD). El módulo de CI permitirá desde compilar la aplicación, hasta publicar las imágenes de Docker y los paquetes NPM de cada una de las partes. Por otro lado, el módulo de CD será el encargado de utilizar esas imágenes que se han publicado previamente y desplegar la aplicación en el entorno correspondiente.

- Entorno orquestado.

  El *pipeline* de CD termina con el despliegue de la aplicación sobre Kubernetes[6], utilizando Helm[7]. Esto permite levantar la aplicación en el entorno que le corresponda, según el estado en el que se encuentra cada versión.

- Análisis comparativo

  Finalmente, se realiza un análisis de las ventajas que ofrece Dagger frente a métodos convencionales.

1.2. Relación de la documentación

- Capítulo 1: Introducción.

  Este capítulo, en el cual se describen la finalidad del proyecto, las tecnologías a utilizar, de manera breve; y la estructura, a grandes rasgos, del trabajo en sí.

- Capítulo 2: Estado del arte y fundamentos teóricos.

  En el segundo capítulo se detallan los conceptos más importantes de CI/CD. Además, se estudia la evolución de las herramientas DevOps[8], incluyendo Dagger como una de las últimas y más innovadoras herramientas en este sector, y se compara con las otras tecnologías.

- Capítulo 3: Diseño y arquitectura del sistema.

  Aquí se describen las tecnologías utilizadas para implementar la aplicación *dummy*. También se explica cómo se ha organizado la infraestructura de despliegue.

- Capítulo 4: Implementación del *pipeline* con Dagger.

  Aquí se indican los pasos que se han dado para crear los *pipelines* con Dagger, utilizando el SDK para definirlos como código. Este es el núcleo del proyecto.

- Capítulo 5: Pruebas y resultados.

  En este capítulo se presentan las pruebas que se han llevado a cabo. Se habla de las dificultades que se han tenido, así como de las ventajas que ofrece Dagger frente a otras tecnologías, aportando comparaciones cuantitativas y cualitativas.

- Capítulo 6: Conclusiones y líneas futuras.

  Finalmente, se resumen los hechos que se han obtenido, se valora el resultado final del uso de Dagger y se indica si ha cumplido con las expectativas. Además, se añaden puntos de mejora o extensiones del proyecto.

Capítulo 2

Estado del arte y fundamentos teóricos

Antes de empezar a escribir código, se deben entender los conceptos fundamentales que permitirán llevar a cabo este trabajo.

Lo que se intenta mejorar utilizando Dagger es el ciclo completo de CI/CD de una aplicación. Por lo tanto, es fundamental definir los conceptos de *Continuous Integration* y el *Continuous Delivery*. Una vez se comprenda a qué se refieren esos términos, se podrán entender los métodos y tecnologías convencionales que permiten implementar dichos procesos. Será entonces cuando se pueda introducir Dagger, un método innovador para realizar *pipelines*, el cual aporta muchas ventajas y se comparará con otras tecnologías disponibles en el sector.

2.1. CI/CD

CI/CD son las siglas de *Continuous Integration*/*Continuous Delivery*, o en casos más específicos, este último también se puedo conocer como *Continuous Deployment*.

Se trata de un conjunto de pasos automatizados, utilizados en el desarrollo de software para llevar el código desde su implementación inicial hasta el despliegue de la aplicación. Estos pasos incluyen:

- Integración de cambios en el código.
- La compilación de la aplicación con los cambios realizados.
- Realización de pruebas.
- Creación y publicación de imágenes de Docker y paquetes NPM.
- Despliegue de la aplicación (modelo *push*[9])

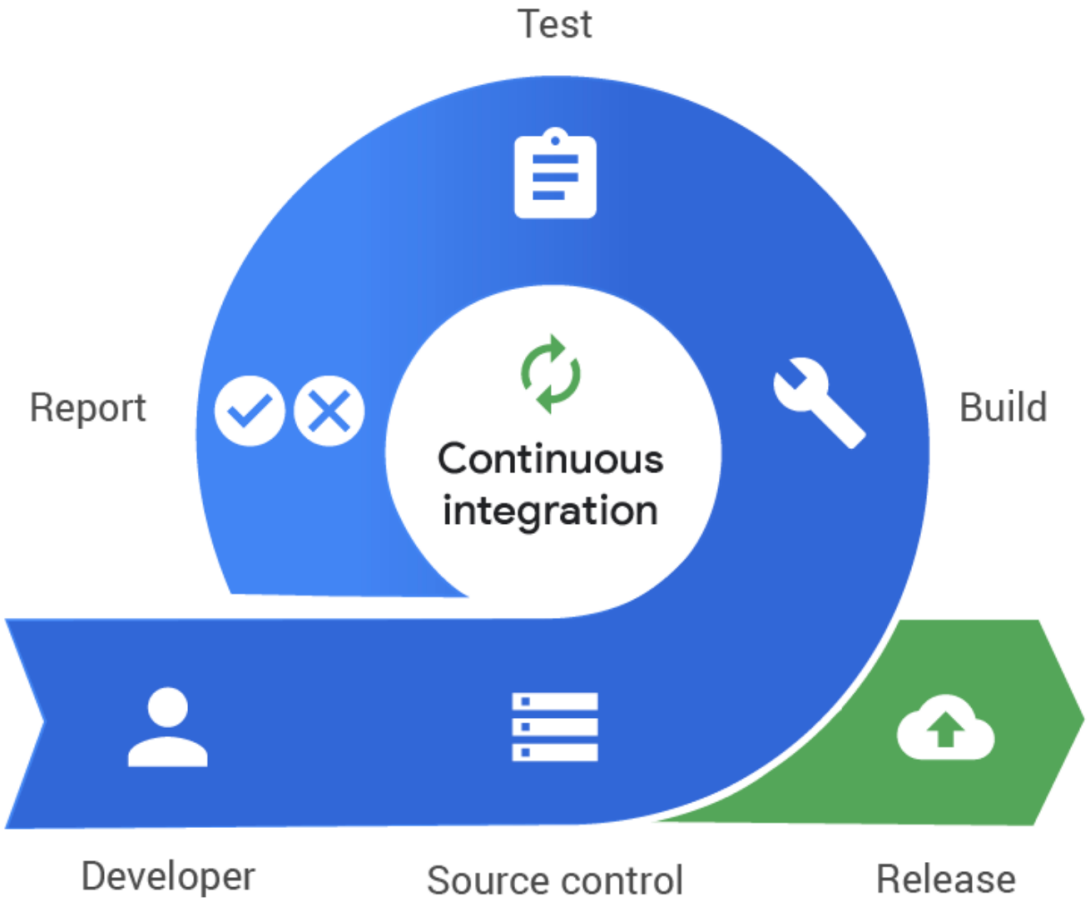

Figura 2.1: Proceso de integración continua.[3]

Continuous Integration

Se basa en la integración de código de manera constante, día a día, en un repositorio compartido por programadores. Cada uno de los programadores realiza cambios en el código y lo integra en el repositorio. Una vez se realizan cambios, estos deben pasar una serie de pruebas para que se incluyan definitivamente en el código fuente de la aplicación (Fig. 2.1).

2.2. Métodos convencionales

2.3. Dagger

Capítulo 3

Deseño

Debe describirse como se realiza o Sistema, a división deste en diferentes compoñentes e a comunicación entre eles. Así mesmo, determinarase o equipamento hardware e software necesario, xustificando a súa elección no caso de que non fose un requisito previo. Debe achegarse a un nivel suficiente de detalle que permita comprender a totalidade da estrutura do produto desenvolvido, utilizando no posible representacións gráficas.

Capítulo 4

Probas

Plan de probas (con evidencias) que verifica a funcionalidade e correctitude global do sistema, e se leva a cabo.

Capítulo 5

Exemplos (eliminar capítulo na versión final)

5.1. Un exemplo de sección

Esta é *letra cursiva*, esta é **letra negrilla**, esta é letra subrallada, e esta é `letra curier`. Letra tiny, scriptsize, small, large, Large, LARGE e moitas más. Exemplo de fórmula: $a = \int_o^\infty f(t)dt$. E agora unha ecuación aparte:

$$S = \sum_{i=0}^{N-1} a_i^2. \tag{5.1}$$

As ecuaciones se poden referenciar: ecuación (5.1).

5.1.1. Un exemplo de subsección

O texto vai aquí.

5.1.2. Otro exemplo de subsección

O texto vai aquí.

Un exemplo de subsubsección

O texto vai aquí.

Un exemplo de subsubsección

O texto vai aquí.

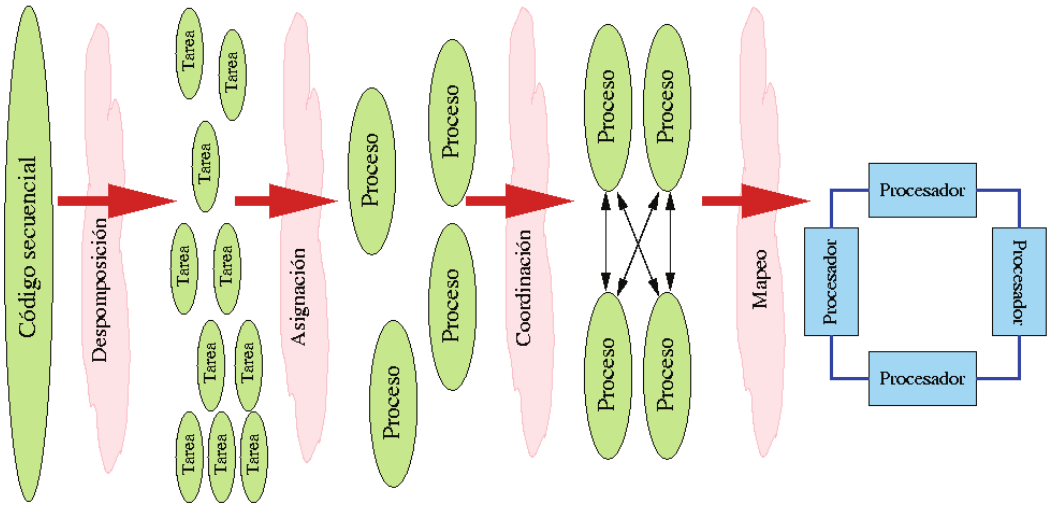

Figura 5.1: Esta é a figura de tal e cal.

Izquierda	Derecha	Centrado
ll	r	cccc
llll	rrr	c

Cuadro 5.1: Esta é a táboa de tal e cal.

Un exemplo de subsubsección

O texto vai aquí.

5.2. Exemplos de figuras e cadros

A figura número 5.1.
O cadro (taboa) número 5.1.

5.3. Exemplos de referencias á bibliografía

5.4. Exemplos de enumeracións

Con puntos:

- Un.
- Dous.
- Tres.

Con números:

1. Catro.
2. Cinco.
3. Seis.

Exemplo de texto verbatim:

```
O texto        verbatim
    se visualiza tal
           como se escribe
```

Exemplo de código C:

```c
#include <math.h>
main()
{   int i, j, a[10];
    for(i=0;i<=10;i++) a[i]=i; // comentario 1
    if(a[1]==0) j=1; /* comentario 2 */
    else j=2;
}
```

Exemplo de código Java:

```java
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); // Display the string.
    }
}
```

Capítulo 6

Conclusións e posibles ampliacións

O traballo describe o grao de cumprimento dos obxectivos. Posibles vías de mellora.

Apéndice A

Manuais técnicos

En función do tipo de Traballo e metodoloxía empregada, o contido poderase dividir en varios documentos. En todo caso, neles incluirase toda a información precisa para aquelas persoas que se vaian encargar do desenvolvemento e/ou modificación do Sistema (por exemplo código fonte, recursos necesarios, operacións necesarias para modificacións e probas, posibles problemas, etc.). O código fonte poderase entregar en soporte informático en formatos PDF ou postscript.

Apéndice B

Manuais de usuario

Incluirán toda a información precisa para aquelas persoas que utilicen o Sistema: instalación, utilización, configuración, mensaxes de erro, etc. A documentación do usuario debe ser autocontida, é dicir, para o seu entendemento o usuario final non debe precisar da lectura doutro manual técnico.

Apéndice C

Licenza

Se se quere pór unha licenza (GNU GPL, Creative Commons, etc), o texto da licenza vai aquí.

Bibliografía

[1] Dagger.io. (2024). Official Dagger Documentation. Recuperade de *https://docs.dagger.io/*.

[2] Fowler, M. (2024). Continuous Integration. Recuperado de *https://martinfowler.com/articles/continuousIntegration.html*.

[3] PagerDuty, Inc. (2025). What is Continuous Integration? Recuperado de *https://www.pagerduty.com/resources/devops/learn/what-is-continuous-integration/*.

[4] Fowler, M. (2019). Software Delivery Guide. Recuperado de *https://martinfowler.com/delivery.html*.

[5] Nx (2025). Everyting you need to know about monorepos, and the tools to build them. Recuperado de *https://monorepo.tools/*

[6] Los autores de Kubernetes (2025). Orquestaciónn de contenedores para producción. Recuperado de *https://kubernetes.io/es/*

[7] Helm authors (2025). The package manager por Kubernetes. Recuperado de *https://helm.sh/*

[8] Atlassian (2025). ¿Qué es DevOps?. Recuperado de *https://www.atlassian.com/es/devops*

[9] Mohammed Nasser (2024). Push vs. Pull-Based Depoyments. Recuperado de *https://dev.to/mohamednasser018/push-vs-pull-based-deployments-4m78*