Continuous Delivery: ¿Qué es y cómo afecta a nuestra arquitectura?

Integrantes:
- Samuel Rodríguez Ares (UO271612)
- Alba Guerrero García (UO266007)
- Victoria Álvarez Sordo (UO266618)

1. ¿Qué es Continuous Delivery?

Continuous Delivery, también denominada como *Entrega continua*, es una práctica propia del desarrollo de software que consiste en la **integración sostenible de todo tipo de cambios** (véase, por ejemplo, nuevas características y correcciones de errores) de una forma **rápida y segura**.

La entrega continua puede ser considerada como una **ampliación de la *integración continua***; mientras que esta última **automatiza** el proceso de compilación y pruebas tras incluir cambios en un repositorio, la entrega continua extiende el proceso desplegando el código con todos esos cambios hacia un entorno de producción; de esta manera, los desarrolladores **siempre disponen de algo** que haya sido probado y tan solo queda en mano de estos tomar la decisión de lanzar una actualización de dicho producto.

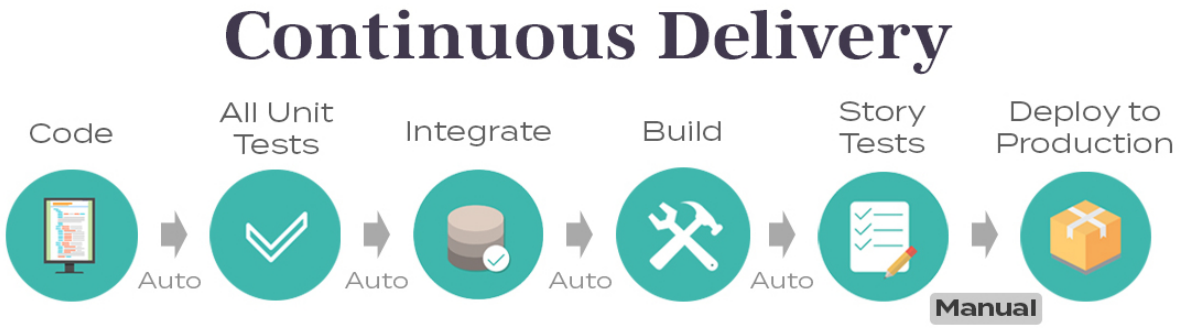

Imagen 1. Recuperado de: DZone

Dado que se trata de un proceso automático, gracias a la entrega continua los **desarrolladores pueden programar el lanzamiento de nuevas versiones** del producto en diferentes períodos de tiempo de acuerdo con los requisitos del cliente.

El hecho de realizar entregas de la forma más frecuente posible ayuda a crear un **mejor entorno de retroalimentación** cada vez que se producen ciertos cambios, de forma que sea posible atajar cualquier impedimento y haciendo así el proceso de entrega más rápido, óptimo y seguro.

Una práctica similar a esta es el *Continuous Deployment*, con la principal diferencia de que el producto es desplegado en producción automáticamente sin intervención humana en caso de que todas las pruebas unitarias y de aceptación sean completadas con éxito.

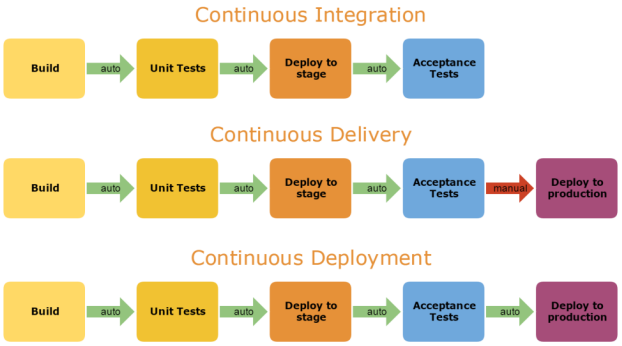

Imagen 2. Recuperado de: CodeProject

2. Principios del Continuous Delivery

- **Una construcción de calidad.** La realización de pruebas automatizadas permiten detectar y corregir defectos antes de introducir una nueva versión; esto resulta mucho menos costoso para el equipo de desarrollo y evita procesos de inspección y pruebas manuales.
- **Trabajo en períodos breves.** La obtención de un feedback temprano supone una gran serie de beneficios, puesto que facilita la solución de problemas y aumenta la eficiencia del equipo de desarrollo, además de resultar económicamente más viable.
- **Distribución inteligente del trabajo.** Delegar en las máquinas aquellas tareas más simples, repetitivas y de menor valor para permitir a los humanos centrarse en actividades más complejas, como la resolución de problemas.
- **Mejora continua.** La intención de mejora debe estar implícita en el desarrollo de cualquier proyecto, sin conformarse con el estado actual del mismo.
- **Responsabilidad conjunta.** Todo el equipo de desarrollo debe velar por la calidad y la estabilidad del software a construir.

3. En qué se basa Continuous Delivery

- **Gestión de configuración.** Existen dos objetivos principales: la reproducibilidad (creación de nuevos entornos a partir de la misma base de forma automática) y la trazabilidad (determinar fácilmente qué dependencias y componentes tiene cada versión del entorno).
- **Integración continua.** El trabajo de los diferentes desarrolladores es combinado en la rama principal de forma frecuente para someter el código a diferentes pruebas, tanto antes como después de la fusión.
- **Prueba continua.** Ejecución de múltiples tipos de pruebas manuales y automáticas a lo largo del proceso de entrega (aceptación, usabilidad, pruebas unitarias…).

El conjunto de estos constituye lo que se conoce por ***deployment pipeline***, un patrón clave en la entrega continua. De esta manera, cada vez que se produce algún cambio, se genera una compilación que es desplegada de forma automática en un entorno de pruebas y los desarrolladores obtienen un feedback directo sobre el resultado de dichas pruebas.

4. Beneficios para la arquitectura

- **Automatización del proceso de publicación.** El equipo es capaz de probar y preparar cualquier cambio que puede ser dirigido a producción de forma automática, mejorando la eficiencia.
- **Liberación de responsabilidades.** Los desarrolladores se eximen de la responsabilidad de llevar a cabo las tareas manuales mencionadas anteriormente, favoreciendo la calidad del código.
- **Mejoras en la depuración.** Disponer de pruebas automatizadas y exhaustivas permite al equipo de desarrollo encontrar los errores de software más fácilmente.
- **Rapidez de entrega.** El cliente recibe actualizaciones con mayor frecuencia, pudiendo solicitar un artefacto de software preparado y que haya pasado las pruebas en cualquier momento.

5. Bibliografía y referencias

- *Amazon Web Services, Inc. (2021). Introducción a AWS CodePipeline.* https://aws.amazon.com/es/devops/continuous-delivery/
- *Atlassian. (2021). Continuous integration vs. continuous delivery vs. continuous deployment.* https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment
- *Humble, J. (2017). What is Continuous Delivery? - Continuous Delivery. Continuous delivery.* https://continuousdelivery.com/
- *Pearson Education. (2010, 7 septiembre). Continuous Delivery: Anatomy of the Deployment Pipeline | Introduction | InformIT. Informit.* https://www.informit.com/articles/article.aspx?p=1621865