Entrega Continua

Andrés Fernández González(UO264699)
Rocío Cenador Martínez (UO226650)

¿Qué es la entrega continua?

La entrega continua es la capacidad para producir software o para incluir cambios como funcionalidad nueva, cambios de configuración o soluciones de errores en el software ya existente en cortos periodos de tiempo, de manera segura y sostenible. Previene que lleguen *releases* a producción que no hayan sido aprobadas y testeadas exhaustivamente. Mitiga los posibles problemas debido a la incorrecta interacción imprevista de los componentes del sistema y su entorno. Al ser un proceso automatizado, el despliegue es rápido, repetible y confiable. Se normaliza el evento de hacer un *release* ya que se realizan más frecuentemente, evitando el estrés de introducir un bug crítico ya que es fácil volver a una versión anterior.

Beneficios

La entrega continua, siempre que se lleve a cabo con la disciplina necesaria para mejorar día a día, proporciona los siguientes beneficios:

• Los cambios que se realicen en el software no afectarán a las demás partes del proyecto ya que son implementaciones sencillas, de manera que puedan hacerse en el momento que se requieran. Esto supone una optimización del tiempo de trabajo, ya que las personas implicadas en el proyecto podrán realizar su parte del trabajo, independientemente de lo que haga el resto del equipo.

• El cliente podrá ir viendo los cambios que se producen antes de llegar al producto final, por tanto, se podrán solucionar posibles errores o realizar cambios que el cliente vea necesario al momento, dando lugar a un software de mejor calidad y que se ajuste a las necesidades del cliente. Esto también supone una reducción en el coste, ya que no tendrán que realizarse grandes cambios en el software.

• La comunicación permanente y fluida entre el cliente y el equipo de desarrollo no sólo son beneficiosas para el resultado del producto final sino que también generan un mejor ambiente de trabajo en el día a día. Todos los agentes implicados en el proceso obtienen por un lado, acceso a las cosas que necesitan cuando las necesitan, y por otro, visibilidad del proceso de manera que resulta fácil identificar, optimizar y eliminar los cuellos de botella que se producen durante la entrega de un software.

• Automatizar el proceso de despliegue permite que sea testeado con mayor regularidad y por lo tanto reducirá el riesgo del *release*. Un software que ha llevado demasiado tiempo ponerlo en un entorno similar al de producción puede resultar en software *undeployable*. Tiende a ser *buggy*

software porque el ciclo de feedback entre el equipo de desarrollo y el de pruebas y operaciones es muy largo.

Principios

La entrega continua cuenta con los siguientes principios:

• Construir calidad: Se refiere a la capacidad de resolver los problemas de inmediato, es decir, la comunicación permanente con el cliente permite identificar software que no satisface las necesidades del cliente y modificarlo al momento, teniendo así un software de mayor calidad y ahorrando tiempo y recursos.

• Trabajar en lotes pequeños: Esto hace referencia al pequeño volumen de trabajo que supone cada entrega, y que permite al equipo de desarrollo recibir comentarios sobre su trabajo rápidamente, facilitando la resolución de errores y aumentando la eficiencia.

• Los ordenadores realizan tareas repetitivas, las personas resuelven problemas. Esto quiere decir que los trabajos automatizados se dejan para los ordenadores, mientras que las personas deben dedicarse a tareas de mayor valor.

• Mejora continua, es decir, mejorar día a día con cada entrega que se realice, sin conformarse.

• Todos son responsables. Esto quiere decir que todas las personas implicadas en el proyecto trabajan en la misma dirección para conseguir objetivos a nivel organizacional, en vez de solo preocuparse por su equipo o departamento.

Deployment Pipeline

Una *deployment pipeline* sirve para modelar el proceso de llevar un software desde el control de versiones a las manos de los usuarios. Te permite ver y controlar el progreso en cada una de sus fases. Esta pipeline forma parte de un proceso aún mayor, el que abarca desde la idea de una característica (feature) surgida en la mente del cliente hasta la entrega de la misma en manos del usuario. Todo este proceso completo se puede modelar a través de un Value Stream Map, el cual muestra una timeline con los tiempos requeridos para dos tipos de actividades: las value-adding (que añaden valor) y las non-value-adding como son los tiempos de espera.

El objetivo de definir una *deployment pipeline* es lograr un loop de feedback mucho más rápido en el proceso de entrega de una aplicación. Gracias al feedback conseguimos reducir los tiempos de espera. Por un lado, los equipos de build y operations no tienen que estar esperando tanto por documentación o propuestas de mejoras, los testers no necesitan esperar por una "versión buena" del software. Así mismo, los equipos de desarrollo recibirán los informes de bugs a tiempo, y no semanas después de haber continuado con el desarrollo de otra funcionalidad. Y por último

evitaremos descubrir al final del proceso que la arquitectura de la aplicación no soporta los requisitos no funcionales.

¿Dónde comienza la deployment pipeline? El proceso se inicia con cada cambio que crea un build. Comienza con un commit al sistema de control de versiones, que pasará por las distintas fases de evaluación, cada una de las cuales aporta una perspectiva diferente. El objetivo es eliminar los candidatos a *release* tan pronto como sea posible en el proceso. Un software que falla en algún *stage* generalmente no será promocionado al siguiente.

Fases de la pipeline

The commit stage, Automated acceptance test stages, Manual test stages y Release stage.

Cuando los desarrolladores introducen cambios en su sistema de control de versiones se crea una nueva instancia de la pipeline. En la primera fase, se realizan los test unitarios, el análisis del rendimiento y se crean los instaladores. La segunda fase se lanza automáticamente si se ha completado la primera con éxito. Es la fase de *Tests de aceptación automáticos*. Su objetivo es asegurar que el software cumple con las expectativas del cliente.

Estas dos fases se consideran parte de la *Integración Continua,* que se centra principalmente en aprobar la validez del código a través de test unitarios y de aceptación. Su uso representa una gran ventaja frente a desarrollos que no la utilizan, pero sin embargo no es suficiente, ya que se centra principalmente en el equipo de desarrollo. El output de la integración continua sirve como input de las siguientes fases (proceso de pruebas manuales y siguientes), por lo cual afecta a todo el proceso de entrega. La mayor parte del tiempo que se pierde en un proyecto ocurre durante las fases de pruebas y operaciones.

A partir de esa segunda fase, el proceso se independiza en diferentes ramas: UAT (user acceptance testing), *Capacity testing* y producción. Lo más frecuente es que éstos procesos no se lancen automáticamente si no que cada equipo controle su inicio a través de la ejecución de un script de despliegue para ese entorno.

Lo que sí se automatiza es el despliegue propiamente dicho a los entornos de testing, staging y producción para eliminar en la medida de lo posible los pasos manuales tan propensos a errores. Es lo que se conoce como *Depliegue Continuo*.

Por lo tanto, la Entrega Continua es un concepto amplio que abarcaría otras prácticas continuas, como el desarrollo, integración, tests, despliegue, monitoreo y feedback continuos.

Buenas prácticas en Deployment Pipelines

- Compila los ejecutables una sola vez:

Si se compila repetidas veces en las diferentes fases de la pipeline (durante el commit, en el test de aceptación, luego una vez por cada destino de despliegue, ...) aumentan las probabilidades de introducir alguna diferencia, lo cual puede derivar en errores que alcanzan la fase de producción.

Estaríamos violando dos principios fundamentales, el de conseguir feedback lo antes posible (ya que el proceso de recompilar lleva tiempo, especialmente en sistemas grandes), y el de construir sobre fundamentos correctos (los ejecutables que se despliegan a producción deberían ser exactamente los mismos que pasaron por todas las fases de tests).

Si no son idénticos, si no que los recreamos, corremos el riesgo de introducir algún cambio y que el ejecutable que entregamos sea diferente del testeado.

- Despliega de la misma forma para todos los entornos:

Es esencial utilizar el mismo proceso de despliegue para todos los entornos. Diferencias entre entornos: sistema operativo, configuraciones de middleware, localización de las bases de datos y servicios externos y otra serie de configuraciones. Todo ello necesita ser configurado a la hora de desplegar, pero no quiere decir que tengas que tener un script diferente para cada entorno. Lo ideal es mantener separadas las configuraciones que son únicas para cada entorno (por ejemplo, usando *property files*). ¿Cómo seleccionar entonces qué *property file* utilizar para un entorno concreto? Se puede controlar esta selección a través de una variable de entorno en el script de despliegue.

- Realiza Smoke-test de los despliegues:

Construye un script que automáticamente compruebe que tu software está corriendo, y que todos los servicios de los cuales depende están funcionando correctamente.

- Despliega a una copia de producción:

Un problema muy común es disponer de un entorno de producción muy diferente al de pruebas y desarrollo. Esto genera desconfianza e inseguridad a la hora de pasar a producción. Lo ideal sería que, si lo permite el presupuesto, se pudiera disponer de una copia exacta de producción, asegurándose de que la infraestructura es la misma, así como la configuración del sistema operativo y los datos de la aplicación.

- Cada cambio debería propagarse por la Pipeline:

Las prácticas anteriores a la introducción de la integración continua agendaban las diferentes fases (compilar cada cierto tiempo, pruebas aceptación nocturnas, test de capacidad durante los fines de semana, etc.). La pipeline usa otra estrategia, que es que cada fase debe provocar el inicio de la siguiente inmediatamente después de completarse con éxito.

- Si cualquier parte de la pipeline falla, páralo todo

Si el despliegue a un entorno da un error, la responsabilidad es de todo el equipo, por lo tanto deberían parar con el resto de tareas hasta que lo arreglen.