CONTINUOUS DEPLOYMENT

Martín Garay, Manuel Frana, Levian Lastra, Lucas Perazzi, Facundo Raviolo, Naim Saadi
Universidad Tecnológica Nacional, Facultad Regional Córdoba
Cátedra Ingeniería de Software
4K1 2019 Grupo 2

RESUMEN: *El avance de las metodologías agiles de desarrollo de software por sobre las metodologías tradicionales en los últimos años, dio lugar al surgimiento de nuevas estrategias que facilitan y automatizan el mismo. Estas estrategias de Desarrollo Continuo de Software, forman un conjunto con ciertas dependencias entre sí, siendo en fin etapas, de las cuales la etapa final es el Continuous Deployment. Este aplica y contiene al Continuous Integration y a Continuous Delivery dentro de su proceso, y de su aplicación obtenemos que los desarrolladores deban generar código de calidad todo el tiempo; los usuarios del negocio recibirán funcionalidades nuevas requeridas, en una cantidad de tiempo mínima, y una rápida corrección de errores en producción. La clave para el funcionamiento correcto de esta estrategia, es la Automatización.*

PALABRAS CLAVE: Automatización, Continuo, Errores, Intervención Humana.

1. INTRODUCCIÓN

Continuous Deployment es una estrategia de entrega de Software basada en testing automatizado. Cualquier código que pase las pruebas se implementa automáticamente en un ambiente de producción.

Esta estrategia, forma parte de un conjunto de prácticas continuas: Continuous Integration, Continuous Delivery y Continuous Deployment, las cuales tienen dependencias entre sí para su correcta utilización.

Hoy en día, esta estrategia es utilizada por una gran cantidad de empresas de Desarrollo de Software, como por ejemplo Google, Facebook, LinkedIn y Netflix, las cuales dependen del Continuous Deployment para realizar la implementación de los cambios realizados en su código. Estos cambios son recibidos por sus usuarios, logrando un rápido feedback por parte de los mismos.

2. CONTEXTO

La industria del Software necesita adaptarse a un mercado que cambia rápidamente. Los procesos tradicionales de desarrollo, como el proceso de Cascada, actualmente ya no son la mejor opción, debido a que estos son propensos a errores, debido a la lenta entrega de productos, la poca comunicación con el cliente, entre otros aspectos.

Las metodologías ágiles, tales como Kanban, Scrum, XP, entre otras, surgieron para resolver estas problemáticas, y su popularidad va en constante crecimiento.

Con las prácticas continuas, es posible realizar entregas de características nuevas del Software, con una mayor frecuencia, correspondiéndose de manera directa con uno de los principios fundamentales de las metodologías ágiles: “La manera más eficiente de satisfacer al cliente, es a través de entregas tempranas y continuas de software con valor”.

Otra ventaja de estas prácticas se centra en que la integración de pequeñas características es menos propensa a causar conflictos, que la manera tradicional de integrar grandes cambios, menos frecuentemente y de manera simultánea.

Si se pudiese resumir en una palabra a Continuous Integration, Continuous Delivery y Continuous Deployment, esta sería *Automatización*. Todas ellas se basan en la automatización de los procesos de testing y despliegue. Se desea minimizar o incluso eliminar la intervención humana, reduciendo a su vez los riesgos de cometer errores y logrando que estos procesos sean más simples y rápidos, al punto de que, en una aplicación adecuada de estas estrategias, cualquier integrante del equipo de desarrollo podría encargarse de estos procesos.

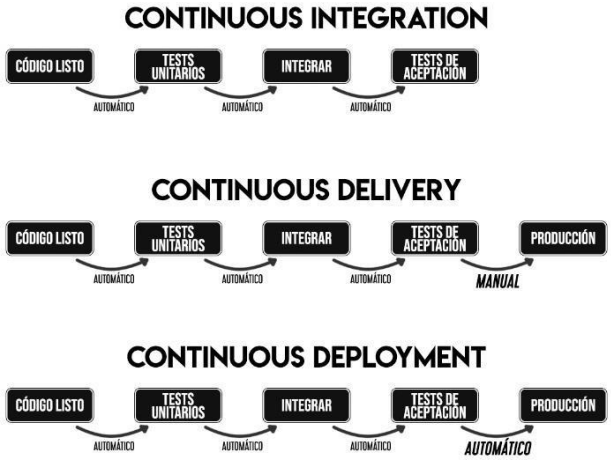
3. OTRAS PRÁCTICAS CONTINUAS

Como anteriormente dijimos, el Continuous Deployment, forma parte de un conjunto de Prácticas de Desarrollo de Software Continuas, las cuales son Continuous Integration, y Continuous Delivery.

Continuous Integration: Es una práctica de desarrollo que obliga a los desarrolladores a integrar su código a un repositorio compartido varias veces al día. Cada integración de código es luego verificada por una serie de pruebas automatizadas, permitiéndole al equipo a detectar problemas de manera temprana. Ya que al integrar el código al repositorio de manera frecuente resulta más fácil detectar y corregir los errores.
“Continuos Integration no corrige errores, pero si los hace dramáticamente más fáciles de encontrar y remover” [1].
El Continuous Integration nos permite tener una visión en tiempo real del estado del Software y de sus medidas de calidad correspondientes.

Continuous Delivery: Es un proceso que se ubica sobre el Continuous Integration, en el cual el código se actualiza de manera constante, y éste debe estar listo para ser implementado. Se deben realizar pruebas para cada pieza de código que sufrió un cambio, para poder validar que el Software esté funcionando correctamente. Según Martin Fowler, *"se está utilizando Continuous Delivery correctamente cuando:*

- *El Software esta en condiciones de ser desplegado a lo largo de todo su ciclo de vida.*
- *El equipo prioriza mantener el software listo para ser desplegado, por sobre crear nuevas características y funcionalidades.*
- *Se puede tener un rápido feedback automatizado, respecto a la preparación para la producción de su Software, cada vez que alguien lo modifica "*



Como puede verse reflejado en el gráfico, la característica de Continuous Integration es que una vez que se tiene el código listo, se procede automáticamente a realizar los tests unitarios. Si estos tests son superados, se pasa automáticamente a la etapa de integración y por último, también automáticamente, todo el producto pasa a los tests de aceptación.

La metodología de Continuous Delivery contiene el siguiente paso: para entregar los nuevos cambios a los clientes más fácilmente, acelera el proceso de despliegue, acortando sustancialmente los tiempos entre cada entrega.

Por último, la estrategia de Continuous Deployment va aún más allá: su objetivo es que no exista intervención humana a la hora de realizar el despliegue, este se produce de manera automática y los tiempos entre cada despliegue son mucho menores.

4. CONTINUOUS DELIVERY VS. CONTINUOUS DEPLOYMENT

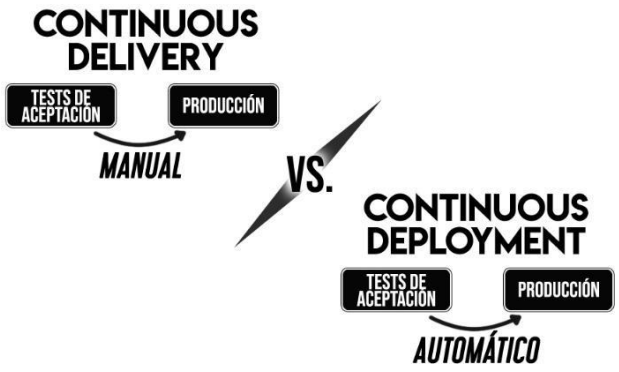
Cuando hablamos de implementar Continuous Deployment, esto implica directamente al Continuous Delivery, pero no a la inversa.

En el Continuous Delivery cada vez que una porción de Software ya esté listo y haya pasado por las pruebas rigurosas y automatizadas para verificar que se comporte como se esperaba, el momento de ponerlo en producción, se deja en manos de las decisiones empresariales y no será definido por el personal de IT, lo cual es distinto para el Continuous Deployment.

En esta práctica, todo el código listo realizado por los desarrolladores, el cual pase las pruebas automatizadas, será puesto automáticamente en producción.

De esta manera, se evita la intervención humana previa a la implementación de los cambios. En Continuous Delivery el despliegue se dispara manualmente, e incluso a veces se realiza de manera completamente manual.

Podemos decir que el Continuous Deployment es el paso siguiente al Continuous Delivery.



Diferencias en la última etapa de desarrollo

5. DE DELIVERY A DEPLOYMENT

Para migrar desde Continuous Delivery hacia la utilización de Continuous Deployment es necesario en primera medida asegurar que las bases de Integración Continua sean correctas y estén correctamente definidas.

Para dar el salto a Continuous Deployment se debe establecer una alta prioridad a la corrección de errores en las versiones de la rama principal, ya que de lo contrario se quedaría expuesto a los mismos riesgos que antes de implementar esta estrategia: los cambios se acumulan en el entorno de desarrollo y los desarrolladores no pueden estar seguros de que su trabajo esté terminado debido a que no saben si sus cambios pasaron los tests de aceptación.

La cobertura de testing es otro de los pilares de esta estrategia, debido a que la utilización de herramientas para este propósito, nos permite asegurar que el Software es correctamente testeado. Se debe intentar una cobertura ideal del 80%, teniendo en cuenta la calidad de las pruebas realizadas.

Otro aspecto importante es la utilización de Monitoreo en tiempo real, ya que debemos contar con formas eficientes de ser alertados en caso de que ocurra algún error durante la implementación automática de los cambios.

Estrictamente relacionado al monitoreo, debemos asegurar que dispongamos de capacidades de "Rollback", para poder volver a un estado del Software en el cual se encontraba anteriormente, con lo cual minimizamos los riesgos de daños en caso de que ocurra un error, o en caso de recibir feedback respecto a un cambio y se desee volver atrás.

Un aspecto mas a tener en cuenta, es que se debe intentar minimizar o evitar los inconvenientes a los usuarios al momento de realizar el despliegue, debido a que algunas veces resultaría necesario cerrar o poner en pausa el funcionamiento del sistema durante este proceso. Si los despliegues serán de manera continua, y durante momentos en los cuales el Software esta en su pico de funcionamiento, se deberá tomar ciertas precauciones para no tener que pausar el funcionamiento.

"Como ya mencioné antes, y quisiera enfatizar esto una y otra vez, el Continuous Deployment no siempre es una opción"[5].

6. FACILIDAD PARA ENCONTRAR ERRORES

Como ya sabemos, en el Continuous Deployment cada vez que un cambio es realizado, este se despliega automáticamente en un ambiente de producción. Decimos que este cambio seria desplegado, de todas formas en algún momento.

Mientras más rápidamente se realice el despliegue, mayores son las chances de encontrar errores que hayan sido cometidos hace poco tiempo, siendo estos también más fáciles de corregir que errores que hayan sido cometidos hace mucho tiempo. A su vez, por la misma naturaleza del Continuous Deployment y Continuous Delivery, los errores serán generalmente pequeños dado que las piezas de Software implementadas también suelen ser pequeñas.

Si cada desarrollador, luego de finalizar una pieza de Software e implementarla, recibe en un tiempo muy corto un mensaje de error desde su ambiente de producción reportando un bug dentro de la funcionalidad recién desplegada, esto hará que el bug pueda ser eliminado fácil y rápidamente. En un tiempo relativamente corto el Software estará funcionando sin bugs nuevamente.

7. CASOS DE ESTUDIO

La utilización del Continuous Deployment, si bien puede generar muchos riesgos y necesita de una gran inversión, suele generar grandes beneficios en compañías de diversos rubros y tamaño.

"Vamos a cambiar por completo la forma en que hacemos software y en algún lugar en medio de este proceso de dos meses vas a cruzar un puente y quemarlo detrás de ti "[2] así se refirió Kevin Scott al desafío de migrar de un desarrollo de software más tradicional a uno de Continuous Deployment. El anterior estaba basado en ramas donde cada desarrollador trabajaba de forma casi aislada y al terminar un gran lote de código correspondiente a alguna característica, presionaba para que esta rama de características se fusionara en la troncal, una vez fusionada, la característica nuevamente necesitaría ser probada para asegurarse de que no rompiera ninguno de los otros códigos nuevos registrados al mismo tiempo.

Con esta nueva práctica un desarrollador escribe un nuevo código en pequeños trozos ordenados y discretos y rápidamente verifica cada fragmento en la línea troncal de software compartida entre todos los desarrolladores de LinkedIn. Este código está sujeto a una elaborada serie de pruebas automatizadas diseñadas para eliminar cualquier error. Una vez que el código pasa las pruebas, se fusiona en el tronco y se cataloga en un sistema que muestra a los administradores qué características están listas para lanzarse en el sitio o en las nuevas versiones de las aplicaciones de LinkedIn.

Desde la implementación de estas prácticas, el desarrollo de software de LinkedIn continúo sin problemas, y las acciones se dispararon un 61 por ciento.

Otro ejemplo de una gran compañía que lo utiliza es Instagram. "En Instagram, implementamos nuestro código de backend 30–50 veces al día cada vez que los ingenieros cometen cambios en el master sin participación humana en la mayoría de los casos"[3]. Esto les permitió que los ingenieros avancen muy rápido al no tener que limitarse a algunas implementaciones por día a horas fijas, sino brindándoles la posibilidad de implementar el código cuando lo deseen, y que los commits erróneos se detectan muy rápidamente y se resuelven, lo que evita retrasos significativos para otros cambios no relacionados.

En compañías a menor escala también existen ejemplos favorables de la utilización de esta práctica, como Grailed, en la cual el desarrollo tradicional les funcionó y era útil cuando el equipo de ingenieros era pequeño pero a medida que este creció, se convirtió en un cuello de botella, lo cual fue mejorado gracias al Continuous Deployment. "Ha pasado casi un año desde que hicimos el cambio a Implementación continua y no hemos mirado atrás. Los comentarios de los antiguos

y nuevos ingenieros de Grailed también han sido muy positivos gracias en gran parte a la reducción de la complejidad y el riesgo de la implementación, el conjunto de pruebas rápidas y la capacidad de realizar pruebas en entornos de control de calidad aislados" [4].

8. GREEN-BLUE DEPLOYMENT

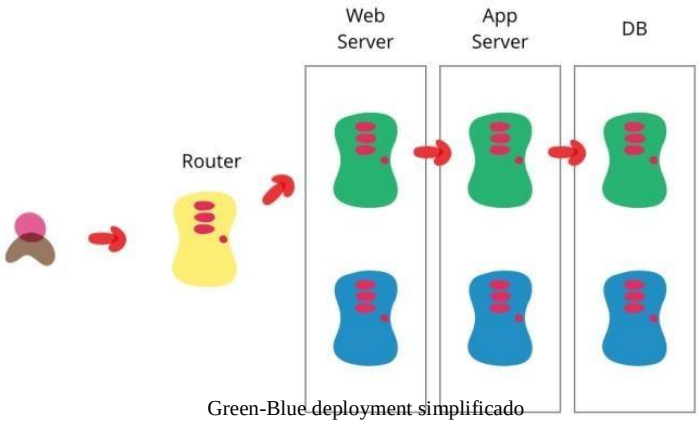
Es un enfoque que consiste en tener dos ambientes de producción muy similar entre sí, disponible en todo momento, lo que permitirá el intercambio entre los mismos para disponer en todo momento de un ambiente donde puedo realizar las pruebas y los testeos, mientras al mismo tiempo se posee un servidor en línea con el producto funcionando.

Esto nos dará la oportunidad de intercambiar de manera inmediata cuando se tienen pruebas exitosas y software funcionando correctamente en aquel servidor que no se encuentra en línea. Esto se puede realizar simplemente redirigiendo el tráfico a aquel servidor que se encontraba en estado de reposo.

La capacidad de realizar un retroceso o "Rollback" resulta muy fácil de llevar a cabo debido a que gracias a este enfoque contamos en todos momentos con dos servidores altamente disponibles en todo momento para ser utilizados como Servidor Base. Además de dar la posibilidad si se desea de utilizar el otro servidor como respaldo.

Estos dos servidores pueden ser diferentes entre sí, pero lo más idéntico posible que se pueda, estos pueden estar compuestos por Hardware distinto o se pueden encontrar situados en un mismo servidor particionado o incluso en máquinas virtuales.

Esta técnica nos da un acercamiento a la total automatización del despliegue propuesta por el Continuous Deployment, pasando directamente desde la última etapa de testing a la producción llevando el tiempo en el que el sistema está fuera de funcionamiento al mínimo y reduciendo los riesgos posibles.



Green-Blue deployment simplificado

10. CONCLUSIÓN

Uno de los principios más importantes que tenemos que recordar del manifiesto de metodologías agiles es que:

"Individuos e interacciones antes que procesos y herramientas"

Y esto resulta muy importante a la hora de entender que implicaría para un equipo u organización implementar Continuous Deployment.

"*Hay un patrón muy recurrente en el desarrollo de software, el cual surge debido a que las personas tratan de copiar el éxito de otros implementando o copiando las herramientas o procesos que aparentan haber dado resultado para dichos equipos, en vez de enfocarse las ideas que originalmente provocaron ese éxito"[5]* – Enuncia el equipo de expertos en software Airplay.

Lo más importante es construir el mejor software que podamos, de la manera más eficiente posible y disponiéndolo a nuestros usuarios de la manera más efectiva que se pueda, mientras a su vez respondemos a las necesidades del negocio y de dichos usuarios.

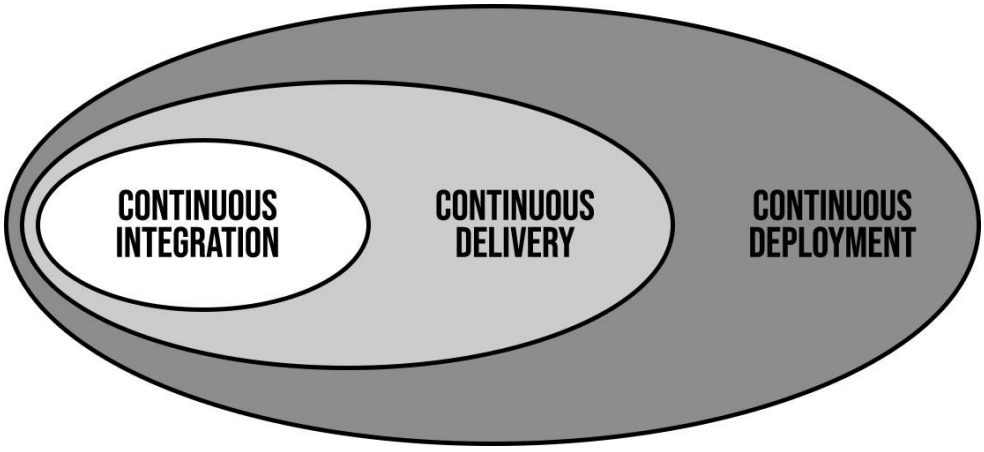
Es por esto que lo importante acá es el *Qué hacemos*, y no tanto el *Cómo*. Uno puede encontrar muchas herramientas o procesos que prometen una mejor implementación de Continuous Deployment, pero al fin y al cabo, lo que uno necesita de manera primordial es un cambio en la manera de pensar y desarrollar, hacerle llegar a la organización lo que se necesita para lograr aplicar esta estrategia de manera correcta, así como también lo que ésta otorga una vez alcanzado. Ya que solo así, con la colaboración y el involucramiento de todas las partes relacionadas, se va a poder alcanzar dicho nivel.

De otra forma, lo único que lograríamos es forzar una manera de trabajo a la que nadie se quiere comprometer, la cual incluso nos obliga a desarrollar más y sin ver ningún beneficio tangible.

Realmente lo que deberíamos lograr es concientizar a cada uno de los miembros de que un buen Continuous Deployment implica que cada uno genere y mantenga código que se encuentre en un estado de "listo para producción" en todo momento, y que todos tienen que comprometerse a construir el mejor software que puedan y de la manera más eficiente y transparente posible. Y que en el evento de que se encuentre un error en producción, no hay que „cortar cabezas" o culpar a responsables, si no entender que nuestro producto se encuentra en un estado de error y que es responsabilidad de todos corregirlo y solucionarlo ya que el equipo entero fue responsable de dicho accionar. Pero como se verá, lograr todo esto resulta muy difícil e incluso confuso.

Por lo tanto, concluimos que hacer Continuous Deployment va más allá de aplicar un proceso y herramientas muy populares que hayan tenido éxito en otras empresas, si no que requiere un compromiso, o más bien como nos gustaría decirle *"un objetivo",* un lugar común e ideal al cual deseamos llegar (un ambiente de desarrollo donde se permita desarrollar código de la mejor manera posible y satisfaciendo la demanda del usuario en el menor tiempo posible sin tener repercusiones graves). Y por esto recomendamos no meterse de lleno o forzar una implementación del Continuous Deployment ya que consideramos que hay etapas previas por las que un equipo u organización necesita atravesar en orden de alcanzar objetivo de utilizar correctamente la estrategia de Continuous Deployment.

Y cuando nos referimos a etapas previas, nos referimos a las prácticas de desarrollo de software continuo en si, como lo son el Continuous Integration y el Continuos Delivery, ya que el Continuous Deployment es la etapa final dentro de las prácticas continuas.



REFERENCIAS

[1] Martin Fowler – Pionero de las practicas continuas de desarrollo de Software.

[2] Kevin Scott, 2013, Marzo, Recuperado de https://www.wired.com/2013/04/linkedin-software-revolution/

[3] Equipo de Ingeniería de Instagram, https://instagram-engineering.com/continuous-deployment-at-instagram-1e18548f01d1#.m25q2a13c

[4] Jose Rosello, Recuperado de https://medium.com/grailed-engineering/continuous-delivery-at-grailed-4268ed9d494f

[5] Sander Rossel, de su libro Continuous Integration, Delivery and Deployment, Editorial Packt, 2017

[6] Equipo de expertos en software Airplay. en su artículo de *„Continuous Deployment for practical people"*.

FUENTES

- "*Continuous Integration, Delivery and Deployment*" - SANDER ROSSEL
- *"Linkedin Software Revolution"* - https://www.wired.com/2013/04/linkedin-software-revolution/
- *"Continuous Integration"* - https://www.thoughtworks.com/continuous-integration
- *"Continuous Delivery VS Continuous Deployment"* - https://continuousdelivery.com/2010/08/continuous-delivery-vs-continuous-deployment/
- *"Continuous Integration in DevOps"* - https://insights.sei.cmu.edu/devops/2015/04/continuous-integration-in-devops.html
- *"Continuous Delivery, Martin Fowler"* - https://martinfowler.com/bliki/ContinuousDelivery.html
- https://martinfowler.com/articles/continuousIntegration.html
- *"Continuous Deployment for practical people"* - https://www.airpair.com/continuous-deployment/posts/continuous-deployment-for-practical-people
- *"Linkedin Software Revolution "* - https://www.wired.com/2013/04/linkedin-software-revolution/
- *"Continuous deployment at Instagram"* - https://instagram-engineering.com/continuous-deployment-at-instagram-1e18548f01d1#.m25q2a13c
- *"Continuous Delivery at GRAILED"* - https://medium.com/grailed-engineering/continuous-delivery-at-grailed-4268ed9d494f
- *"Green-Blue Deployment"* - https://lostechies.com/gabrielschenker/2016/05/23/blue-green-deployment/
- *"Green-Blue Deployment "* - https://martinfowler.com/bliki/BlueGreenDeployment.html