Universidad Tecnológica Nacional
Facultad Regional Córdoba
Ingeniería de Software
Grupo N°5

Continuous Deployment y algunas Estrategias de Despliegue

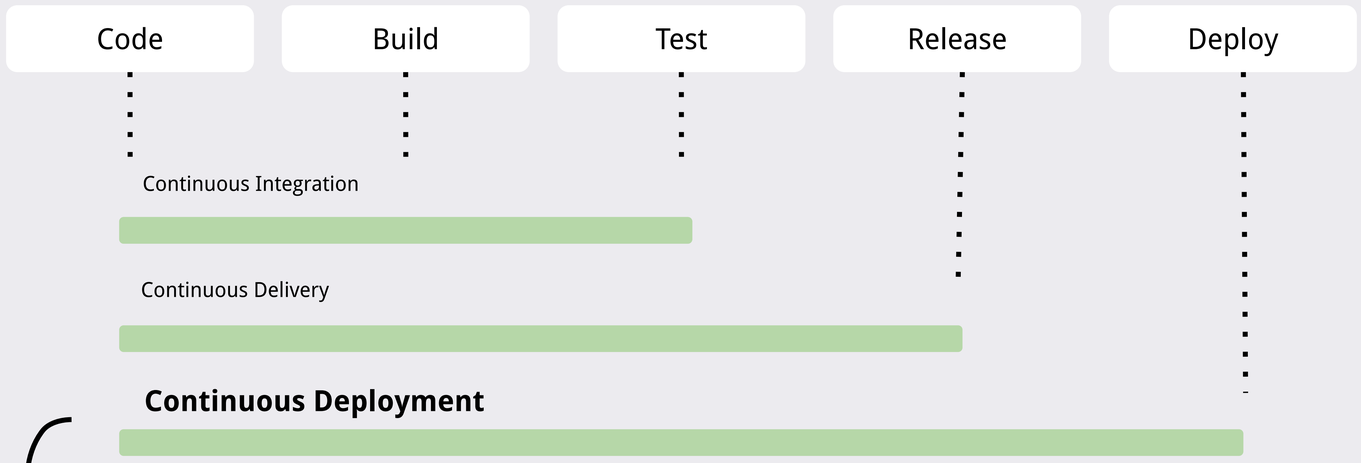
Introducción

La puesta en producción de un producto de software puede ser sumamente intimidante, y ni hablar si pensamos en automatizar este proceso.
Justamente de esto se trata **Continuous Deployment**: automatizar la validación de cambios y la verificación de que sean estables, para desplegarlos en un entorno de producción de forma inmediata. Si bien parece una locura (y no siempre se puede implementar), cientos de empresas emplean **CD** en sus procesos de desarrollo de software, y no es sin justificación. Automatizado o no, el despliegue debe llevarse a cabo, y para ello distinguimos 6 estrategias distintas, cada una con sus pros y contras.

Es Fundamental...

Saber que, para implementar **Continuous Deployment**:

1. Es **necesaria** una buena implementación de **Integración Continua**, garantizando que el nuevo código compila y tiene una calidad razonablemente buena.
2. El conjunto de tests automatizados debe ser **robusto y suficiente**.
3. Al momento de decidir si implementar o no **CD**, hay que considerar no solo aspectos técnicos, sino también del **negocio**.
4. Debemos utilizar una **Estrategia de Despliegue** que permita fácilmente **deshacer** un despliegue, en caso de bugs o fallos.



- **Mejora de calidad del software**
- **Mayor velocidad de desarrollo**
- **Más fácil de aprender para los usuarios**

¿Vale la pena?

Si bien cada proyecto debe evaluarse de forma particular y sabiendo que la principal desventaja de **CD** es el alto costo inicial (además de requerir mantenimiento para su correcto funcionamiento), éste tiene claros **beneficios**:

1. En caso de encontrar un bug o fallo, es más fácil recordar lo que desarrollamos ayer que lo que desarrollamos hace 3 meses. Esto lleva a una:
2. **Mejora de calidad del software**. Esto también se debe a que cada release es lo más pequeña posible (cada commit es un release), lo que facilita encontrar la raíz de los problemas.
3. **Aumenta la velocidad de desarrollo.** Tener releases pequeños permite a los desarrolladores trabajar de forma más eficiente, pudiendo concentrarse en contextos más pequeños. El punto anterior contribuye también a este beneficio.
4. **Más fácil de aprender para los usuarios.** Es mucho más sencillo para las personas aprender con **cambios pequeños y frecuentes** (e incluso quizá ni los noten) que con cientos a la vez.

Materiales y Métodos

La investigación realizada para el desarrollo de este póster se basó en el material sugerido por la cátedra para los temas en cuestión: ***Continuous Integration, Delivery and Deployment***, por Sander Rossel. Por otro lado, nos valimos de distintos artículos web, cuya validez fue verificada de la siguiente forma:

1. Comprobar que el contenido del artículo esté en línea con el material dispuesto por la cátedra.
2. Validar la experiencia y estudios del autor, comparando el breve resumen de sus perfiles en el artículo con sus perfiles de Linkedin.
3. Comprobar que los artículos no sean contradictorios entre sí.

Conclusiones

A pesar de su alto costo inicial y el esfuerzo de mantenimiento que requiere, el **Despliegue Continuo** es una excelente herramienta para optimizar el proceso de desarrollo de software, aumentar la calidad de nuestro producto, experimentar minimizando riesgos y mantener a nuestros clientes felices. Según cual sea nuestro objetivo, vamos a optar por una u otra **Estrategia de Despliegue**, recordando que las aptas para **Continuous Deployment** son aquellas que permitan deshacer los últimos cambios realizados velozmente y sin mayores dificultades.

Ya sea que automaticemos o no el proceso de despliegue, hay que llevarlo a cabo. Para ello, distinguimos las siguientes estrategias:

- **A/B Testing:** la versión **B** se libera a una porción de usuarios, bajo una condición específica.

- **Sombra:** la versión **B** se libera al mismo tiempo que la versión **A**, y recibe el mismo tráfico que **A**, sin impactar la performance. Una vez validada la estabilidad de **B**, se despliega definitivamente.

- **Blue/Green**: la versión **B** se despliega junto a la versión **A**, luego el tráfico se traslada a la versión **B**.

- **Canary:** la versión **B** se libera a una porción de usuarios, luego se libera al resto de los usuarios.

- **Recrear:** apagamos la versión **A** y luego desplegamos la versión **B.**

- **Incremental:** la versión **B** reemplaza lentamente a la versión **A.**

Referencias

1. Sander Rossel. *Continuous Integration, Delivery and Deployment.* **2017**.
2. Sten Pittet, s.f., ingresado 3 de Septiembre de 2020, <https://www.atlassian.com/continuous-delivery/continuous-deployment>.
3. Dan Quine, **2016**, ingresado 3 de Septiembre de 2020, <https://medium.com/continuous-delivery/why-continuous-deployment-matters-to-business-6a79b5602145>.
4. Etienne Tremel, **2017**, ingresado 4 de Septiembre de 2020, <https://thenewstack.io/deployment-strategies>.
5. IBM Cloud (**2019**) *Continuous Deployment vs Continuous Delivery*. Disponible en: https://www.youtube.com/watch?v=LNLKZ4Rvk8w