Prácticas Continuas - Despliegue Continuo en el Desarrollo de Software

Bosio Franco, Cerezo Facundo, Henry Guido, Strasorier Ariel, Uanini Gabriel, Viotti Franco

Ingeniería de Software, 4K2,
Facultad Regional Córdoba,
Universidad Tecnológica
Nacional, Córdoba, Argentina

bosiofranco94@gmail.com

facundomcerezo@gmail.com

GuidoHenry63@gmail.com

ariel.strasorier@gmail.com

gabrieuanini@gmail.com

francoviotti@gmail.com

***Abstract*— El siguiente informe fue creado con dos objetivos en mente. El primero, hacer una breve introducción a las prácticas continuas en el desarrollo de software, nombrando las características principales de cada una de las prácticas existentes y ventajas que conlleva su utilización a los resultados de un proyecto, para luego hacer foco sobre la práctica de Despliegue Continuo, realizar un análisis detallado de sus características y aplicación en proyectos de software.**

El segundo objetivo está directamente ligado a la existencia de este informe. La lectura, comprensión y confección de este tipo de documentos son una parte importante del desarrollo profesional y académico de un Ingeniero ya que los mismos son un recurso importante para la publicación y difusión de conocimiento. Comprender cómo se confeccionan y poder elaborarlos son habilidades que todo profesional del área debe poseer.

I. Introducción

El aumento de competitividad en el mercado de desarrollo de software trajo como consecuencia que las empresas deban prestar especial atención y asignar recursos para llevar a cabo el desarrollo y entrega de software funcional a ritmos cada vez más acelerados.

Las prácticas continuas son algunas de las técnicas enfocadas a dar apoyo a organizaciones, con el objetivo de disminuir los tiempos de desarrollo y entrega de características de software funcional, evitando perder la calidad de estas.

Dentro de las prácticas continuas nos encontramos con la Integración Continua (Continuous Integration), la cual aconseja la integración de trabajos en progreso múltiples veces por día, y por otro lado, se encuentran la Entrega Continua (Continuous Delivery), donde en la etapa de Distribuir el Incremento de Producto de Software realizado se realiza de forma manual, consumiendo tiempo de los miembros del equipo que en el caso de implementar prácticas de Despliegue Continuo (Continuous Deployment) dicha etapa se ejecuta de manera automática, liberando completamente al equipo de desarrollo para que pueda continuar con las tareas que se encuentran en espera.

La correcta utilización de prácticas continuas provee una serie de beneficios al desarrollo de software, la retroalimentación provista, tanto por el proceso de desarrollo como por el cliente, es mayor y más rápida. Por otra parte, las entregas frecuentes, constantes y confiables logran aumentar la confianza de los clientes y la calidad del producto.

II. desarrollo

A. Despliegue Continuo en el Desarrollo de Software

La extendida adopción de principios Lean y metodologías ágiles en el desarrollo de software han puesto en evidencia la necesidad y ventajas de la flexibilidad y capacidad de adaptación en el mercado de software. Por otra parte, el desarrollo ágil ha establecido algunos de los cimientos para el Despliegue Continuo. Por ejemplo, algunos principios ágiles referencian directamente al Despliegue Continuo, “Nuestra principal prioridad es satisfacer al cliente a través de la entrega temprana y continua de software de valor” y “Entregar con frecuencia software que funcione, en periodos de un par de semanas hasta un par de meses, con preferencia en los

períodos breves". Sin embargo, el desarrollo ágil se ha enfocado principalmente en el proceso de desarrollo a nivel del equipo de desarrollo por medio de prácticas como Programación Extrema y Scrum.

El Despliegue Continuo, en cambio, intenta llegar a una situación ideal en la cual funcionalidades de software son desplegadas de forma continua y automática (sin intervención manual) a los entornos de producción, y, donde la información provista por los usuarios es el motor principal para la innovación. En lugar de trabajar durante meses para realizar una entrega de gran tamaño, con múltiples características de software, las empresas limitan su ciclo de desarrollo de forma sustancial, pasando de plazos de hasta dos meses a periodos cortos de un par de semanas, días o, en casos extremos, horas.

Todos los modelos de despliegue continuo comparten tres temáticas esenciales: Despliegue, Continuidad y Velocidad, por tanto, podemos decir que: En [8] los autores definen al despliegue continuo como: "El Despliegue Continuo es la habilidad de entregar características de software valiosas a usuarios, bajo demanda y a voluntad (Despliegue), en series o patrones, con el fin de conseguir un flujo continuo (Continuidad) y en ciclos considerablemente más cortos que en los métodos tradicionales, limitando dichos ciclos a semanas, días o incluso horas (Velocidad)."

La utilización del Despliegue Continuo implica que todo cambio que ha sido previamente testeado es desplegado de forma automática a producción, resultando en una gran disminución en los tiempos de desarrollo y en la eliminación tareas manuales, el proceso se encuentra mayoritariamente automatizado, sólo una prueba fallido puede prevenir que un cambio sea desplegado a producción. Es decir, permite liberar mayor cantidad de código en menor tiempo sin perder calidad, a la vez que reduce costos y evita regresiones, interrupciones o daños a las métricas clave del negocio.

El Despliegue Continuo afecta directamente a algunos actores del desarrollo de software:

1) Clientes: Realizar despliegues de funcionalidades en el momento de su finalización implica que los clientes podrán acceder a ellas rápidamente, esto aumenta la confianza del cliente y le permite observar y analizar, en forma activa, los cambios en el sistema, esto logra una incluirlo en el desarrollo del producto.

2) Desarrolladores: Esta práctica resulta en menor presión sobre el equipo de desarrollo, al no existir un día de lanzamiento (Release Day) semanal/mensual los desarrolladores no necesitan hacer un esfuerzo excesivo para lograr finalizar las funcionalidades establecidas en la planificación de reléase (release planning), si el programador requiere horas extra para finalizar una característica específica entonces la misma será liberada unas horas más tarde y no deberá posponerse hasta la siguiente entrega.

3) Gerente: Los despliegues continuos permiten a los gerentes realizar un seguimiento detallado del desarrollo del producto y les permite analizar el progreso de este.

4) Administradores de Sistema: Por una parte, la automatización del proceso reduce la cantidad de tareas manuales realizadas por los administradores de sistemas, por otro lado, la inclusión de características pequeñas y discretas permiten facilitar la detección de errores o defectos que afectan al sistema de forma negativa.

La implementación de este enfoque no debe tomarse a la ligera ya que es una tarea compleja y sumamente costosa, la capacidad de realizar casos de prueba debe ser altamente efectiva, ya que la calidad de estos afectará directamente a los productos desplegados, también se debe asegurar que la capacidad de generar documentación pueda mantener el ritmo de entregas constantes. Uno de los costos más significativos está relacionado a la necesidad de instalar y mantener un servidor de integración continua que permita realizar el monitoreo de repositorios y llevar a cabo testeos automáticos.

B. Diferencias entre Entrega Continua y Despliegue Continuo

Mientras que la integración continua, la entrega continua y el despliegue continuo tienen el mismo objetivo en común, que es lograr que el desarrollo y entrega de nuestro software sea cada vez más rápido y robusto. Se diferencian esencialmente en el alcance de cada uno de ellos.

Como en la Fig. 1, la entrega continua es una serie de prácticas diseñadas para asegurar que el código se pueda desplegar en producción de forma rápida y segura, donde cada cambio se entrega en un entorno similar a producción y logrando que tanto las aplicaciones como los servicios atraviesen rigurosas pruebas automatizadas. Sin embargo, mantiene las decisiones de despliegue como un paso manual.

El despliegue continuo es el siguiente paso de la entrega continua: Cada cambio que logra pasar todas las pruebas automatizadas es desplegado en producción automáticamente. Por lo que el despliegue continuo debe seguir un proceso automatizado.

Hay situaciones de negocio en las cuales no es posible llevar a cabo la automatización del despliegue debido a que se debe completar alguna funcionalidad previamente. Mientras que el despliegue continuo puede no ser adecuado para todas las compañías, la entrega continua es un requerimiento básico para las correctas prácticas de DevOps.

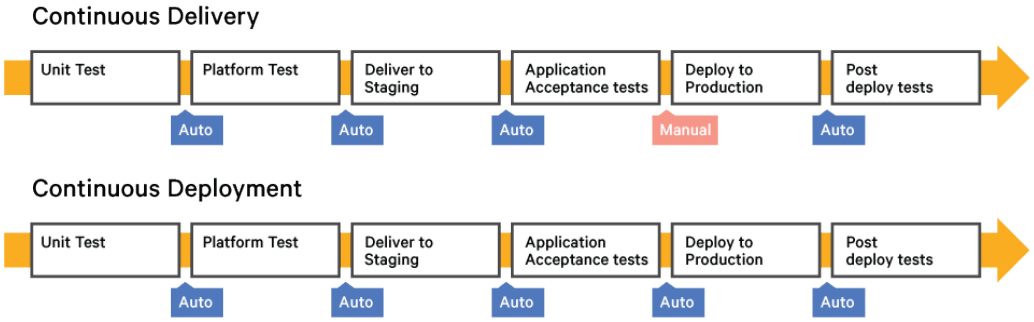

Fig. 1 Entrega continua vs Despliegue continuo

C. *Automatización del Despliegue Continuo*

Es común que, al trabajar en grandes y complejas aplicaciones, éstas deban ser desplegadas en múltiples entornos antes de ponerse en producción. Esto requerirá una gran cantidad de tiempo y esfuerzo para lograr que funcionen correctamente en todos los entornos, donde también será necesario solucionar aquellos problemas que puedan ocurrir al integrar nuestra aplicación con otros sistemas. Éste trabajo puede ser aún más complejo si los procesos de construcción, prueba y despliegue se realizan manualmente.

Es por esto que es clave que todo este proceso se lleve a cabo automáticamente, lo que ayudará a ahorrar tiempo del equipo de desarrollo y a identificar de manera temprana problemas con el despliegue.

Además, la automatización del despliegue nos será de utilidad ya que los scripts de despliegue puedan ser utilizados como la documentación de todo este proceso, evitando tener otra clase de documentos o manuales que necesiten ser actualizados junto con el proceso de despliegue.

Sin embargo, la automatización de todas estas tareas es una actividad compleja que no está libre de desafíos. En [4] Jez Humble, Chris Read y Dan North definen cuatro principios esenciales para el proceso de automatización:

1) Entregar software funcionando en cada etapa de construcción: No debe haber etapas separadas para artefactos intermedios que surjan a consecuencia del desarrollo modular de nuestro sistema. Conviene que el framework del producto se desarrolle y evolucione al mismo ritmo que los demás componentes que lo vayan a utilizar.

2) Desplegar los mismos artefactos en cada entorno: Nuestra compilación debe generar uno o más artefactos que puedan ser desplegados. Estos artefactos deben ser los mismos en todos los entornos, ya sea en el entorno de desarrollo, entorno de preparación y entorno de producción. Hacer esto nos permite probar lo que se lanzará finalmente al entorno de producción. Bajo este principio entra en juego la Gestión de Configuración del Software (SCM, Software Configuration Management), ya que uno de sus principales objetivos es la de lograr ejecutar la aplicación en entornos nuevos tan rápido y simple como sea posible.

3) Automatizar las pruebas y el despliegue: En la etapa final de la línea de despliegue se automatizan las pruebas del sistema y el despliegue de este. Existen 3 conceptos para tener en cuenta que dictarán cómo lo pondremos en práctica:

- Tener pruebas independientes entre sí y capaces de dar una respuesta binaria (pasa o falla).
- Automatizar de manera completa el despliegue en cada entorno en el que trabajemos.
- Realizar pruebas de entorno mediante pruebas de humo o smoke tests, para comprobar que nuestro producto funcione efectivamente sobre el mismo y no entre en incompatibilidades o conflictos.

El objetivo de estas actividades es que el equipo de desarrollo sea capaz de configurar un entorno, desplegar el sistema y probarlo de manera automática, intentando reducir al mínimo el tiempo malgastado.

4) Evolucionar la línea de despliegue a medida que la aplicación se desarrolle: Consiste en adaptar nuestra línea de despliegue según el avance del proceso de desarrollo. Este principio es importante debido a que el software cambia constantemente, con lo que definirlo de manera completa antes de haber escrito es un desperdicio (ya que es muy probable que lo tengamos que cambiar).

D. *Pasos para la Implementación del Despliegue Continuo*

Eric Ries plantea los siguientes para llevar a cabo una correcta implementación del despliegue continuo [6]:

1) Paso 1: La mejor manera para poder asentar una correcta base para la implementación del despliegue continuo es la de una inversión suficiente como para poder hacer frente al costo que conlleva un servidor adecuado y estable. En dicho servidor de integración continua (Continuos Integration) se ejecutarán todas las pruebas automáticas pertinentes como las pruebas unitarias, pruebas funcionales, pruebas de integración, etc. De esta forma en un solo lugar podemos ejecutar y monitorizar las pruebas por cada commit realizado. Se agregará una nueva prueba automatizada en caso de arreglar un error de esta forma podremos subsanar futuros errores en las mismas

situaciones.

2)Paso 2: La siguiente pieza de infraestructura que se necesita es la implementación de otro servidor el cual se encargará de las tareas de control del código fuente. Su papel en la implementación continua es la de detener cualquier código que me cause errores, mal funcionamiento, falla, etc. al proceso de integración continua, prohibiendo el paso del commit que contiene la porción del código nuevo al repositorio. De esta manera nos encontramos con la primera retroalimentación de la implementación continua, desarrollamos tan rápido como se pueda, pero con calidad, si desarrollamos más rápido caeremos en un error ya que producto de ese aceleramiento estaremos propensos a cometer errores que posteriormente nos ocupará más tiempo para solventar.

3) Paso 3: Las implementaciones que se realicen tendrán que ser, idealmente, de igual forma. Para lograr este objetivo se tendrá que obtener un proceso formal, claro y explícito a seguir, el cual debe estar al alcance de todos los miembros del equipo. En el momento en el que se tenga que subir el código a producción se deberá proceder únicamente con la utilización de este proceso previamente elegido, obteniendo así implementaciones idénticas unas con otras. En un principio lo ideal es que este proceso sea manual y sencillo para que todos puedas utilizarlo sin mayores problemas, pero a medida que se avanza y se adquiera experiencia, paulatinamente se volverá un proceso automático.

4) Paso 4: No importa que tan bueno sea el proceso de implementación los errores siempre pueden aparecer , por ello se necesita de una plataforma de monitoreo que informe de tales fallos, tales notificaciones se enviarán a los responsables directos, de tal manera que se presente para poder participar en la depuración correspondiente analizando la situación por la cual se capturó el mal funcionamiento del sistema, permitiendo continuar con la línea de producción del sistema previa solución del problema.

5) Paso 5: Por último, se utiliza una técnica muy potente llamada los cinco “porque”, esta técnica nos permite impulsar potenciales mejoras al proceso de desarrollo de forma incremental. Con esta técnica lo que se busca es intentar llegar a la causa raíz del problema, preguntando recursivamente a sí mismo el ¿por qué? de allí su nombre. Esta técnica puede ser utilizada en despliegue continuo de forma tal que aplicando solo una sencilla regla podremos sacarle gran ventaja, la regla consiste en que cada vez que hagamos un análisis a raíz del problema en estudio, realizar una inversión proporcional en prevención en cada uno de los niveles que se descubran con cada pregunta, sin discriminar el tamaño del problema. Dicha inversión no debería superar el costo del problema analizado, es por ello que hablamos de proporcional. Como resultado se obtendrá que el sistema se pueda adaptar a las circunstancias particulares que se puedan presentar y normalmente no se encuentran contempladas, ahorrándonos tiempo y esfuerzo, los cuales se pueden invertir en otros aspectos más productivos.

E. Despliegues Azul y Verde

Según Jez Humble y D. Farley, dentro de las técnicas más importantes para el despliegue de software es tener dos versiones idénticas del entorno de producción, que se denominan azul y verde [3].

En el ejemplo en la Fig. 2, cuando se desea poner en producción una nueva versión del software, se despliega en el entorno azul. No afectando de ninguna manera el entorno verde. En el entorno azul se pueden correr pruebas de humo para comprobar que todo funcione correctamente. Cuando todo esté en orden y las pruebas hayan corrido de forma exitosa, se cambia la configuración del router y se apunta al entorno azul. Entonces el entorno azul se vuelve la nueva versión en producción en menos de un segundo.

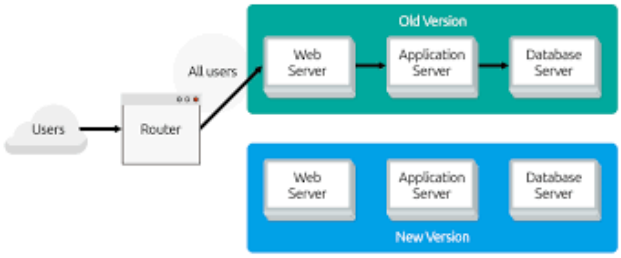

Fig. 2 Despliegue azul y verde

Esta técnica nos permite que de forma muy rápida si algo llegara a funcionar mal, se puede volver al entorno verde, por lo que es necesario tener un plan de rollback.

Es recomendable además que todo este proceso se realice en los horarios nocturnos o de menor carga.

F. Despliegue Canario

Cuando se quiere lanzar nuevo código en el producto de software, pero también reducir el riesgo de algún error, es cuando se utiliza el despliegue canario. De este modo, nos aseguramos de estar enviando una prueba que podamos lanzar a todos los usuarios una vez que estemos seguros de que funciona correctamente. Sólo se trata de elegir al canario. Como en la Fig. 3, por comodidad muchos eligen al 5% de su tráfico. De esta forma, si algo llegara a fallar, únicamente ese 5% se vería afectado. También es posible incluir otros canarios con diferente alcance de tráfico; así se incrementa la confianza en los nuevos cambios.

Luego de que estemos seguros de que los cambios funcionan correctamente, se puede hacer el despliegue para el total de los usuarios. En este caso ya no sería necesario el canario, ya que sólo sirvió para tener certeza de que todo marchaba correctamente.

Sin importar si el producto de software es una aplicación web o móvil, los despliegues de canario funcionan en los dos casos. Otra ventaja es que pueden adaptarse a distintas infraestructuras, ya sea en la nube o una infraestructura híbrida; incluso, se puede utilizar con contenedores.

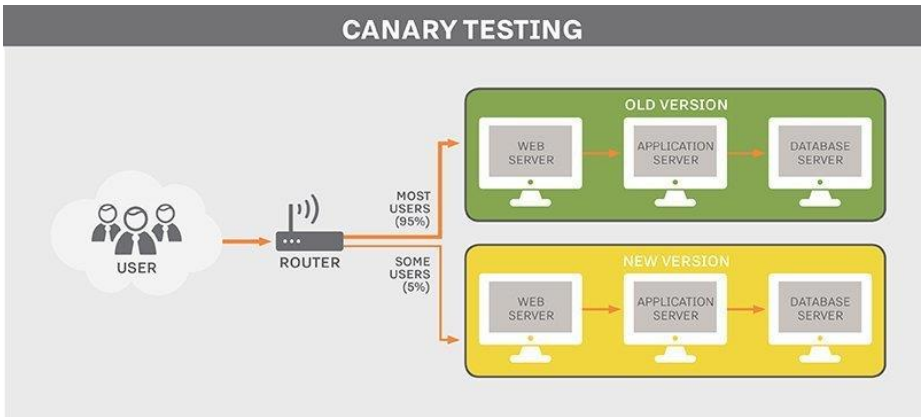

Fig. 3 Despliegue Canario

G. Herramientas Involucradas

La implementación de un esquema de despliegue continuo utiliza de manera casi imprescindible un conjunto de herramientas a disposición del equipo de desarrollo. Muchas de estas se utilizan también en la Integración Continua y la Entrega Continua:

1) Sistema de Control de Versiones: el cual maneja las versiones de los artefactos que conforman el producto. El más utilizado y de naturaleza descentralizada es Git, escrito por Linus Torvalds, creador de Linux. Otras opciones son Subversion, Mercurial y Perforce

2) Servidor de Automatización / Integración Continua: utilizado para la gestión de las actividades automatizadas que caen dentro de la Integración Continua y la Entrega Continua, como las tareas y las recopilaciones periódicas del software. Es la herramienta principal del despliegue continuo, ya que permite realizar pruebas, integrar el código automáticamente y conectarse con el resto de las herramientas de una manera centralizada. Como ejemplos tenemos Jenkins, Bamboo de Atlassian y Azure DevOps de Microsoft.

3) Software de Automatización de Pruebas (Test Automation): se utiliza para realizar pruebas, ya sea en un proceso de prueba ya definido o para hacer pruebas adicionales que serían difíciles de hacer de otra manera. Como ejemplo de esto tenemos las pruebas de GUI, que consisten en simular acciones de un usuario en una aplicación gráfica para comprobar que lo obtenido sea lo esperado. Una de las herramientas de testing más conocidas es Selenium, orientada a aplicaciones web.

4) Sistema de Automatización de Despliegue (Deploy Automation): herramienta utilizada para automatizar el despliegue. Permiten definir cómo se va a desplegar el software, bajo qué eventos (por ejemplo, la integración de un branch de código, o cada intervalo de tiempo) y a qué entornos (desarrollo, staging, producción, etc.). Como ejemplo tenemos DeployBot, Octopus Deploy para aplicaciones .NET y Elastic Beanstalk de Amazon Web Services.

III. CONCLUSIONES

Como conclusión de este Informe Técnico hemos arribado qué uno de los más grandes beneficios que se le pueden otorgar a una empresa de desarrollo de software, es la capacidad de discernir entre si es o no necesario aplicar prácticas de Entrega Continua y posteriormente Despliegue Continuo.

Refiriéndonos a capacidad de discernir como el efecto de darnos cuenta si el proyecto en el que estamos trabajando requiere que se apliquen estas prácticas o no es realmente necesario. Ya sea porque el negocio no demanda este conjunto de prácticas de automatización o porque el tamaño del sistema no es lo suficientemente grande para aprovechar las ventajas.

Uno de los puntos claves que debería quedar en claro después de leer con atención este informe técnico, es la diferencia entre Entrega Continua (Continuous Delivery) y Despliegue Continuo (Continuous Deployment), básicamente los dos conceptos son técnicas de automatización bastantes complejas que logran que el desarrollo y entrega de nuestro software sea cada vez más rápido y robusto.

La diferencia sustancial entre uno y el otro, es que en la Entrega Continua la etapa de Distribuir el Incremento de Producto de Software realizado se realiza de forma manual, esto consume tiempo de los miembros del equipo que en el caso de implementar prácticas de Despliegue Continuo la etapa en cuestión se ejecuta de manera automática, liberando completamente al equipo de desarrollo para que pueda continuar con las tareas que se encuentran en espera.

REFERENCIAS

[1] "Formato IEEE Para Presentar Artículos." [Online]. Available: http://www.unisecmexico.com/archivosPDF/Formato_IEEE.pdf.

[2] Rossel Sander, *"Continuous Integration, Delivery and Deployment"*, Editorial Packt, 2017

[3] Jez Humble and D. Farley, *Continuous delivery*. Addison-Wesley, 2011.

[4] Jez Humble, Chris Read, and Dan North. 2006. *The Deployment Production Line. In Proceedings of the conference on AGILE 2006 (AGILE '06).* IEEE Computer Society, Washington, DC, USA, 113-118. DOI: https://doi.org/10.1109/AGILE.2006.53

[5] *"Continuous Delivery Vs. Continuous Deployment: What's the Diff?"* Puppet, 15-Feb-2015. [Online]. Available: https://puppet.com/blog/continuous-delivery-vs-continuous-deployment-what-s-diff.

[6] E. Ries, *"Continuous deployment in 5 easy steps - O'Reilly Radar,"* Oreilly.com, 2009. [Online]. Available: http://radar.oreilly.com/2009/03/continuous-deployment-5-eas.html

[7] Shahin, Mojtaba & Ali Babar, Muhammad & Zhu, Liming. (2017). *Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices.* IEEE Access. PP. 10.1109/ACCESS.2017.2685629.

[8] P. Rodriguez, A. Haghighatkhah, L. E. Lwakatare, S. Teppola, T. Suomalainen, J. Eskeli, T. Karvonen, P. Kuvaja, J. M. Verner and M. Oivo. *"Continuous deployment of software intensive products and services: A systematic mapping study."* Journal of Systems and Software, vol 123, pp. 263-291, 2017.

[9] *"Continuous integration vs. continuous delivery vs. continuous deployment"*, atlassian, [online]. Available: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment

[10] *"Practical continuous deployment: a guide to automated software delivery"*, atlassian, 17-feb-2017, [online]. Available: https://www.atlassian.com/blog/continuous-delivery/practical-continuous-deployment

[11] *"How to get to Continuous Deployment"*, atlassian, [online]. Available: https://www.atlassian.com/continuous-delivery/continuous-deployment