Integración continua

Arguello, Matias Nicolas
Facultad Regional Cordoba
Universidad Tecnológica Nacional
Las Varillas, Cordoba, Argentina
matiarguello011@gmail.com

Cerutti, Juan Manuel
Facultad Regional Cordoba
Universidad Tecnológica Nacional
Cordoba, Argentina
juanmanuelcerutti@gmail.com

Heinzmann, Ignacio
Facultad Regional Cordoba
Universidad Tecnológica Nacional
Cordoba, Argentina
ignacio.heinzmann@gmail.com

Romero, Facundo
Facultad Regional Cordoba
Universidad Tecnológica Nacional
Chos Malal, Neuquen
facurome26@gmail.com

Risso, Uriel Nicolas
Facultad Regional Cordoba
Universidad Tecnológica Nacional
Villa Carlos Paz, Cordoba, Argentina
uriel.nr.18.6@gmail.com

I. INTRODUCCION

En el presente paper científico presentaremos una de las tres metodologías utilizadas por los equipos ágiles a la hora de desarrollar un producto de software, en este caso, la integración continua (CI). Aquí abordaremos qué es y en que se basa la integración continua, los pasos que uno debe seguir para el correcto funcionamiento, los beneficios y las prácticas a seguir a la hora de llevar a cabo esta práctica. También abordaremos los diferentes tipos de test que se pueden llevar a cabo ya que la idea principal de la integración continua se basa en el testing automático.

II. RESUMEN

La integración continua(CI) es una práctica para el desarrollo de software donde un equipo de desarrollo integra su código de manera temprana a la rama principal o repositorio de código. El objetivo es reducir el riesgo de tener una “integration hell” por el hecho de esperar hasta el fin del proyecto o del sprint para fusionar el trabajo de todos los desarrolladores.

Las principales ventajas que tiene la implementación de CI son las de ahorrar tiempo a la hora de encontrar y corregir errores durante la etapa de desarrollo al momento de integrar el código, y también brinda un mayor entendimiento del código base y las características que les brinda el sistema desarrollado a los usuarios finales.

III. LA INTEGRACIÓN, EL DESPLIEGUE Y LA ENTREGA CONTINUA

La integración, despliegue y entrega continua son metodologías de trabajo implementadas por los equipos de desarrollo que trabajan en entornos ágiles para poder mejorar su calidad de software en tiempos de desarrollo más reducidos. Estas tres metodologías son implementadas por los llamados DevOps y se relacionan directamente con el principio del desarrollo ágil que enuncia: “La prioridad es satisfacer al cliente a través de releases[1] tempranos y frecuentes”, dado que permiten desarrollar construir y probar nuevos códigos rápidamente a través de la automatización.

En el caso de la integración continua, cuando hablamos de automatización estamos haciendo referencia a lograr establecer pruebas de manera automatica.

Para llevar a cabo la integración continua se necesita automatizar los diferentes tests que se aplicarán en cada cambio que se realice en la mainline[2], también se recomienda aplicarlos en cada rama del repositorio. De esta manera se podrán capturar los problemas en corto tiempo y minimizar desorganizaciones e interrupciones en el equipo de trabajo.

Para ello existen una gran variedad de Tests para implementar, pero no todos son implementados en igual cantidad. Como se puede ver en la pirámide propuesta por Mike Cohn [Figura 1], los tests unitarios son los que mayor importancia se les da, siendo el mayor enfoque que debemos tener al realizar la automatización de pruebas.

Sin entrar en detalle, podemos dar una breve definición de cada uno de ellos:

Tests unitarios: Alcance limitado y por lo general, verifican el comportamiento de métodos o funciones individuales.

Tests de integración: Se asegura que múltiples componentes se comporten de manera correcta en conjunto. Esto puede involucrar diferentes clases como también la integración con servicios.

Tests de aceptación: Son similares a los test de integración pero ponen su foco en el caso de negocio y no en cada uno de los componentes individualmente.

Tests de interfaz de usuario(UI): Aseguran que la aplicación funcione correctamente desde la perspectiva del usuario

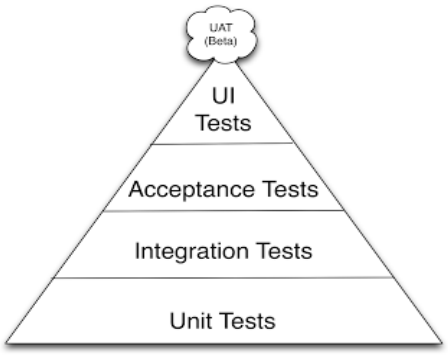


Figura 1: Importancia de los tests

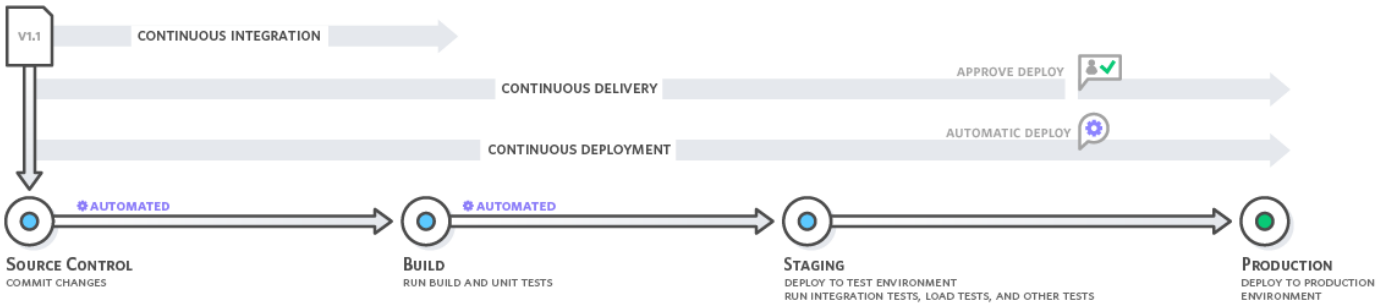


Figura 2: Etapas abarcadas por la integración continua, despliegue continuo y entrega continua

IV. PRACTICAS DE LA INTEGRACIÓN CONTINUA

Si bien la automatización de pruebas es lo más importante, no es lo único que se necesita para llevar a cabo esta metodología. Para ello hay que tener en cuenta las siguientes practicas:

A. *Mantener un repositorio de código único*

Los proyectos de software involucran una gran cantidad de archivos para la construcción de un producto. Por lo que cuando tenemos equipos de trabajos numerosos puede ser complicado llevar un seguimiento del código y de las versiones del producto. Es por eso que se opta por utilizar un repositorio de código donde se almacenen todos los archivos y configuraciones para el desarrollo del producto.

B. *Automatizar una build[3]*

Llevar el código fuente a un sistema funcionando puede tornarse un proceso complicado y demandante de tiempo, se debería automatizar. El objetivo es crear un script que con un comando logre tener el sistema funcionando.

Una buena herramienta para realizar una build analiza qué necesita cambiarse como parte del proceso. Dependiendo de qué haga falta, será de interés tener diferentes cosas en la build. Se puede construir un sistema con o sin código de pruebas, otros componentes se pueden construir aislados. El build script[4] debería permitir la selección de estas alternativas para diferentes casos.

C. *Realizar commits[5] a la mainline todos los días*

Una de las cuestiones más importantes para que esta metodología se ejecute con éxito es cambiar la cultura del equipo para asegurarse que los desarrolladores no trabajen días en una funcionalidad sin realizar fusiones a la rama principal.

Esto incluye que los desarrolladores logren que su código pase todos los test locales para luego hacer un commit a la rama principal.

Al realizar este proceso frecuentemente, los desarrolladores van a poder encontrar rápidamente los conflictos entre los códigos de diferentes desarrolladores. La clave es encontrar y resolver estos problemas rápido ya que si perduran en el tiempo, va a ser más difícil de detectarlos y más aún de resolverlos.

Mientras más frecuente realices commits, menos lugares se tendrán para ver cuando se deba solucionar/encontrar conflictos.

D. *Hacer tu build self-testing*

Para evitar pasar por alto todos los bugs[6] que puede llegar a tener un programa se realizan testings automatizados los cuales durante el proceso de compilación ayudan de manera más rápida y eficiente a detectar dichos bugs para lograr un programa ejecutable sin fallos, el testing no es perfecto pero si permite encontrar una gran cantidad de bugs. El auto testing se debe iniciar desde una pequeña prueba y cualquier error encontrado debe hacer fallar toda la compilación. Las pruebas no garantizan la falta de errores pero ejecutar estas pruebas es mejor que no ejecutar nunca las que sí lo garantizan.

E. *Every Commit Should Build the Mainline on an Integration Machine*

El equipo va a estar constantemente ejecutando commits por ende se realizarán testeos constantemente con lo que se lograra que la mainline se mantenga en buen estado pero esto no se garantiza por diferentes razones, ya sean de disciplina a la hora de ejecutar pruebas o a la máquina de cada desarrollador. Por lo que para asegurar el correcto estado de la mainline se debe lograr que cada compilación se realice en una máquina de integración y solo en caso de que la compilación haya sido satisfactoria se realiza el commit. Como esto es controlado por el desarrollador este tiene que estar pendiente en caso de fallos para solucionarlo. Hay dos formas de realizarlo, ya sea manualmente en donde cada desarrollador inspecciona la integración del código o en un servidor de CI que monitorea el repositorio de datos cada vez que se realiza un commit. El servidor de CI en la llegada de un commit lo compila, realiza las pruebas en la máquina de integración y en caso de encontrar un error envía una notificación al que realizó el commit, de lo contrario, realiza el commit a la mainline.

F. *Arreglar la build rota inmediatamente*

Una parte muy importante para llevar a cabo una build continua es que si la build en el mainline falla, necesita ser arreglada rápidamente, debe ser prioritario. Lo más rápido es revertir la build al último commit de la mainline, llevando el sistema a una build que se conoce que funciona correctamente. El equipo no debería gastar tiempo llevando a cabo el debugging[7] de una mainline rota.

G. *Keep the Build Fast*

El objetivo de la integración continua es brindar una rápida retroalimentación. Ya que requiere realizar testing en cada commit que se realiza, la compilación del mismo no puede ser lenta ya que cada minuto que uno ahorra en la compilación es un minuto que cada desarrollador se ahorra, aunque sea poco tiempo esto termina ayudando para los “commits frecuentes”. Cabe destacar que si realizar una prueba lleva mucho tiempo no se podría pensar en que el proyecto fuese corto. La compilación de los commits es la única parte que debe ser rápida y debido a esto muchas veces se pierde en el testing la capacidad de descubrir más bugs, por lo que se desea lograr

un balance entre tiempo y detección de errores. Cuando esto se logre cada desarrollador puede trabajar con confianza en lo que está haciendo.

H. Probar en un clon del entorno de producción

Si buscamos correr las pruebas en diferentes entornos, diferentes al de producción, vamos a correr el riesgo de obtener diferentes resultados. Por lo cual se busca establecer un entorno de prueba que sea lo más exacto posible al entorno de producción. Usando el mismo software de base de datos, con las mismas versiones, misma versión de sistema operativo, librerías, direcciones de IP, puertos, etc.

Es verdad que existen ciertos límites ya que si estamos desarrollando software para escritorio, cada una de las computadoras tiene diferentes software de terceros por lo que no se puede hacer un clon de entorno para cada una de ellas, sería muy costoso. Pero a pesar de estos límites, el objetivo tendría que seguir siendo el mismo, tratar de duplicar lo más similar posible al entorno de producción aceptando los riesgos de las diferencias que puede haber entre los entornos de test y producción

I. Hacer accesible para cualquier persona la obtención de el último ejecutable

La idea en el desarrollo del software es que este sea correcto por lo que si se deja la última versión del software disponible para cualquier persona está podrá encontrar errores para mejorarlo, por lo que se intenta que la mayor cantidad de personas poseen la última versión. Para ello se vuelca la última versión o la más entendible para los usuarios en un lugar común donde ellos puedan acceder a él en cualquier momento.

J. Todos pueden ver que está pasando

La integración continua está ligada a la comunicación, por lo que debe asegurarse que todos los integrantes del equipo puedan ver fácilmente el estado del sistema y los cambios que se realizan. Una de las cosas más importantes que comunicar es el estado del mainline build. Cuando se utiliza un proceso de CI manual la máquina de integración muestra en todo momento el estado de la mainline, en cambio, cuando se utiliza una página web de los servidores de CI puede proporcionar información acerca de aquel que está construyendo, de los cambios que hicieron y un historial de cambios, lo que permite a los miembros del equipo realizar un seguimiento de la actividad reciente del proyecto.

K. Desarrollo automatico

Para llevar adelante la integración continua se necesitan múltiples ambientes, uno en el cual se pueda llevar a cabo pruebas de commits y uno o más para hacer pruebas secundarias. Se debe automatizar estas tareas. Aunque no se lleve adelante el despliegue en producción todos los días, el realizarlo automáticamente ayuda tanto a acelerar el proceso, como a reducir errores y costos.

V. BENEFICIOS DE LA INTEGRACIÓN CONTINUA

El mayor beneficio de la integración continua es la reducción del riesgo ya que en todo momento se tiene control sobre el proceso, se sabe que funciona, que no, y los errores pendientes que tiene en el sistema.

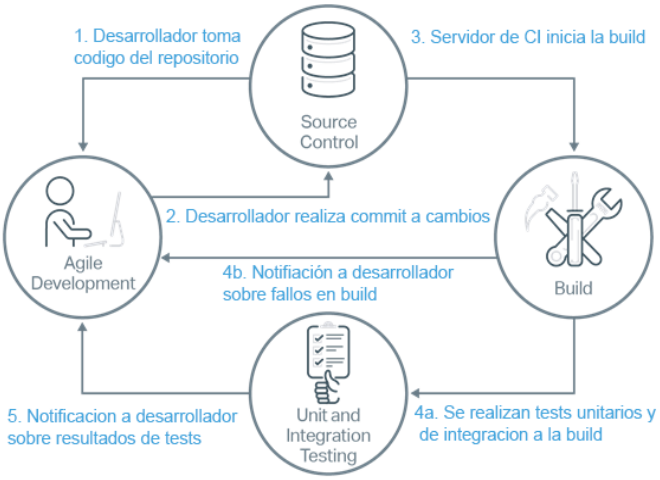
La integracion continua no deshace los errores, pero los hacen muchos más fáciles de encontrar y a su vez de eliminar ya que son detectados en su fase más temprana. Esto debido a que solo ha cambiado una pequeña parte del sistema que es aquella con la que acaba de trabajar el desarrollador por lo que nuevamente hace que sea más sencillo encontrar y deshacerse del error.

La integración continua es barata, el hecho de no integrar continuamente es caro.

La integración continua trae múltiples beneficios a la organización:

- Decirle adiós a largas y tediosas integraciones
- Incrementar la visibilidad permitiendo una mejor comunicación
- Capturar problemas temprano y cortarlos de raíz
- Pasar más tiempo añadiendo nuevas funcionalidades y menos haciendo debugging
- Reduce problemas de integración permitiendo entregar software más rápido
- Deja de esperar para saber si tu código va a funcionar
- Construir una base sólida.

VI. PASOS DE LA INTEGRACIÓN CONTINUA



1) El desarrollador toma el código que necesite en el repositorio para comenzar con su desarrollo de software. Luego al finalizar, realiza test locales.

2) Cuando está listo, se realizan commits a los cambios realizados.

3) El servidor de CI monitorea el repositorio y detecta cuando se quiere hacer un cambio, de ser así dispara la creación de una build.

4) Luego el servidor CI realiza test unitarios y de integración sobre la build creada y procede a realizar las siguientes actividades:

- Asigna un nombre (tag) a la versión de la build que se acaba de crear
- Notifica a los miembros del equipo que se realizó una build exitosa

- Pone a disposición artefactos desplegables para realizar tests

En el caso de que falle la creación, se le notifica al desarrollador y el mismo comienza desde el paso 1 nuevamente con el objetivo de resolver el problema.

5) A este punto, los cambios que han sido corroborados en el paso 2 han sido satisfactoriamente creados y asignado un etiquetamiento a la build creada. De esta forma se finaliza notificando al desarrollador del caso exitoso.

VII. Comparación de herramientas de integración continua

A la hora de implementar la integración continua se cuenta con varias herramientas que se han desarrollado a lo largo del tiempo, pudiendo así darnos la posibilidad de elegir la que más nos convenga ya sea por rendimiento o compatibilidad con alguna otra aplicación que estemos utilizando. Nombraremos las herramientas de integración continua más conocidas del mercado actual. Viendo sus diferentes características para tener una primera aproximación a lo que puede brindar cada una de ellas.

A. *Bitbucket Pipelines*

Es una herramienta de CI directamente integrada en Bitbucket, un sistema de control de versiones en la nube. Bitbucket Pipelines te permite fácilmente habilitar CI si el proyecto ya está en Bitbucket. Las pipelines de Bitbucket se administran como código para que pueda confirmar fácilmente las definiciones de pipelines y comenzar las compilaciones. Bitbucket Pipelines, además ofrece CD. Esto significa que los proyectos construidos con Bitbucket Pipelines también se pueden implementar en la infraestructura

Características:

- Fácil de configurar e instalar
- Experiencia unificada de Bitbucket
- Nube por terceros

Website: https://bitbucket.org

B. *Jenkins*

Es que es un servidor de integración continua de código abierto escrito en Java. Es, de lejos, la herramienta más utilizada para gestionar construcciones de integración continua y canalizaciones de entrega. Ayuda a los desarrolladores a construir y probar software continuamente. Aumenta la escala de automatización y está ganando rápidamente popularidad en los círculos DevOps

Características:

- Es de código abierto
- No requiere instalaciones o componentes adicionales.
- Es gratis
- On-Premise (instalación local, “en casa”)

Website: https://jenkins.io/

C. *AWS CodePipeline*

Amazon Web Services es uno de los proveedores de infraestructura en la nube más dominante del mercado. Ofrecen herramientas y servicios para todo tipo de infraestructura y tareas de desarrollo de código.

CodePipeline es la herramienta de CI que ellos ofrecen. CodePipeline puede interactuar directamente con otras herramientas existentes que provee AWS para proporcionar una experiencia perfecta.

Características:

- Totalmente en la nube
- Integrado con Amazon Web Service
- Soporte de plugins personalizado
- Control de acceso robusto

Website: https://aws.amazon.com/codepipeline/

D. *CircleCI*

CircleCI es una herramienta de CI que se combina de buena manera con Github, uno de las herramientas más populares de sistemas de control de versión hosteados en la nube.Circle es una de las herramientas de CI más flexibles ya que soporta una matriz de sistemas de control de versión, sistemas de contenedores y mecanismos de entrega. CircleCI puede ser hosteado en las instalaciones o utilizarse a través de la nube.

Características:

- Notificaciones desencadenantes de eventos de CI
- Rendimiento optimizado para compilaciones rápidas
- Fácil debugging a través de SSH y compilaciones locales
- Análisis para medir el rendimiento de la compilación

Website: https://circleci.com/

E. *Azure Pipelines*

Azure es una plataforma de infraestructura en la nube de Microsoft, el equivalente al Amazon Web Service pero de Microsoft. Cómo Code Pipeline, Azure ofrece una herramienta de CI que se integra completamente con el conjunto de herramientas de hosteo de Azure.

Características:

- Integrado con la plataforma de Azure
- Soporta la plataforma de Windows
- Soporte de contenedores
- Integración de Github

Website: https://azure.microsoft.com

F. *GitLab*

Gitlab es una nueva herramienta de CI que ofrece una experiencia completa de DevOps. Fue creada con la intención de mejorar la experiencia general de Gitub. Ofrece una moderna experiencia de usuario(UX) con un soporte de contenedores.

Características:

- Hosteo local o en la nube
- Testing de seguridad continuo
- UX fácil de aprender

Website: https://about.gitlab.com

G. *Atlassian Bamboo*

Sin embargo Bitbucket Pipelines es una opción puramente hosteada en la nube, Bamboo ofrece una alternativa de auto-hosteo

Características:

• Mayor integración con el conjunto de productos de Atlassian

• Un amplio mercado de complementos y complementos

• Soporte de contenedores con agentes docker

Website: https://www.atlassian.com/software/bamboo

VIII. Conclusion

Antes que nada, cabe destacar que ninguno de los integrantes del grupo tenía conocimientos previos sobre Integración continua. Sí habíamos escuchado sobre las CCC (Continuous Integration, Continuous Deployment y Continuous Delivery) pero nunca supimos de qué trata cada una. Es por eso que este informe nos fue de gran ayuda. Al tener que decidirnos por una de las tres prácticas, aprendimos un poco de cada una para tomar la decisión y luego detallamos la que nos pareció más interesante de aprender, en este caso, integración continua (CI).

La toma de dicha decisión resultó cuando leímos sobre las prácticas de CI, nos hizo reflexionar en varios aspectos sobre un desarrollo de software en el que los errores que ocurran van a tender, en su mayoría, a ser de bajo impacto porque si logramos establecer una cultura tal que en nuestro equipo implique la realización de commits diariamente, vamos a poder identificar los errores de cada commit con mayor facilidad y su magnitud no va a ser tan grande como si los desarrolladores trabajaran en su código por días y luego realicen un commit, esto lleva a que surjan una gran cantidad de errores muy difíciles de identificar. Es por eso que podemos decir que si logramos una buena implementación de la integración continua vamos a reducir sustancialmente los errores y la velocidad de resolución, con ello se ahorraría así mucho tiempo del desarrollo de software además de lograr un mayor entendimiento del código y del producto final entre el mismo equipo.

En adición a lo anterior queremos destacar cómo esta práctica nos ayuda a reducir el riesgo de tener una “integration hell”, es decir que por el hecho de esperar hasta el fin de proyecto o del sprint para fusionar el trabajo de todos los desarrolladores, se convierta en un “infierno” integrar las partes de cada uno.

A medida que avanzábamos en la integración continua, nos fuimos dando cuenta cada vez más de lo importante que son los repositorios a la hora de realizar un proyecto, si logramos establecer una correcta estructura del mismo con un plan de configuración acorde al proyecto que se vaya a desarrollar, lograríamos mantener la información acomodada y con claridad para poder controlar el versionado de la misma y así no tener problemas a la hora de querer realizar cambios.

Teniendo en claro, ahora sí, qué se necesita para llevar a cabo la integración continua, y habiendo indagado un poco en que se utiliza para llevarla a cabo podemos partir de esta base para mejorar la calidad de nuestros futuros productos en los que trabajaremos, debiendo aprender, eso sí, las demás prácticas que sólo conocimos muy por arriba para poder abordar de la mejor manera este tema.

Glosario

[1] Releases: Nuevas versiones de una aplicación informática.(es un build particular que hacemos público o accesible para ciertos usuarios, o para cierto uso específico.)

[2] Mainline: Rama principal del repositorio. Aquí se introducen las nuevas características y desarrollos del proyecto la cual uno se encuentra completamente seguro que funcionan correctamente.

[3] Build: es la versión "binaria" o ejecutable del artefacto. Probablemente el resultado de haber compilado los archivos fuente y diversas transformación (componer, comprimir, empaquetar, etc.). Es la versión más simple para poder "ejecutar" el software (teniendo en cuenta que también se podría ejecutar desde el entorno de desarrollo, pero requiere más infraestructura, etc).

[4] Build script: código para llevar adelante la creación de una build.

[5] Commit: confirmar un conjunto de cambios provisionales de forma permanente.

[6] Bugs: un error o un defecto en el software o hardware que hace que un programa funcione incorrectamente.

[7] Debugging: Depurar

[8] Third party software: “Software de terceros”. Programas de computadora que son suministrados o desarrollados para un propósito particular por una compañía diferente de la que suministró o desarrolló los programas existentes en un sistema particular

Bibliografia

Max Rehkopf - “What is Continuous Integration” https://www.atlassian.com/continuous-delivery/continuous-integration

Sergi Fernández Cristià - Volviendo a las raíces: integración continua del software (primera parte) - 30/04/2018. Disponible en: https://apiumhub.com/es/tech-blog-barcelona/integracion-continua-del-software/

Mojtaba Shahina , Muhammad Ali Babara y Liming Zhub: “Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices” – 2017. Disponible en: https://arxiv.org/ftp/arxiv/papers/1703/1703.07019.pdf

Martin Flower. “Continuous Integration“– 01/05/2006. https://martinfowler.com/articles/continuousIntegration.html