Hoy vamos a hablar de integración continua desde conceptos hasta realizar una implementación completa.

* **Que problemas resuelve (contar la historia)**
  + Minimizar los problemas de integración
  + Funciona en mi máquina.
* **Que es integración continua**

Es una filosofía en la cuál se busca que todos los miembros del equipo integren su trabajo de manera muy pero muy frecuente. Esta filosofía fue introducida como una práctica recomendada para desarrollar software dentro de la metodología XP ya que permitía ahorrar gran cantidad de tiempo y encontrar los errores de manera oportuna y solucionarlos rápidamente.

* **Practicas** 
  + **Mantener un único repositorio de código**

Hay herramientas que las llaman repositorios, source controls systems, configuration managment toosl, etc, cualquier sea el nombre que tenga, esta herramienta es parte integral de muchos proyectos de software, lo realmente triste es que no de todos. (Preguntar quienes ya usan un repositorio)

Las ventajas:

* + - Poder guardar y coordinar todos los cambios realizados por el equipo, es decir ya no se necesita integrar manualmente, cuando un miembro quiere guardar sus cambios, los hace en ese servidor y cuando otro quiere actualizar solo descarga y se resuelven conflictos en la medida de lo posible.
    - Asimismo mantiene una histórico de todos los cambios, lo cuál nos permite ir adelante o atrás en cualquier momento.
    - Crear variantes del producto. Version, plataforma, ambiente, cliente.

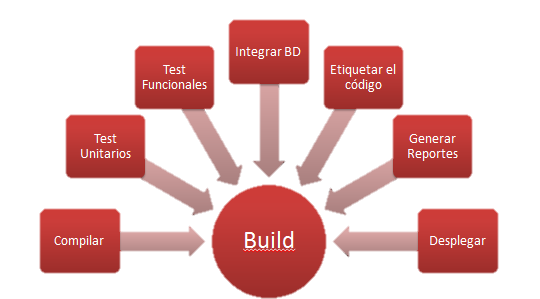
(Creamos el repositorio y subimos el proyecto ahí)

* + **Automatizar el Build**

Antes que nada que es el Build:

El proceso o todos los pasos necesarios de convertir los archivos de código fuente a código ejecutable, *verificar que todo funciona como se espera y que los estándares definidos de manera única por el equipo se han cumplido.*

Preguntar en base a la definición de que tareas está compuesto un build.



No todos los builds son distintos, es más un mismo proyecto puedes tener varios tipos diferentes de builds dependiendo del contexto en el que se esté ejecutando el build: ejm: el build de un desarrollador será un poco diferente al build del producto que se va a pasar a calidad y a producción.

Xq automatizarlo, xq son tareas muy repetitivas q no hacen perder tiempo (productividad) y propensas a error (eliminación de errores humanos), nos ayudan a crear la estructura de nuestro build.

Las herramientas que nos permiten automatizar nuestro build, son las denominadas “build scripts”. (Maven, MSBuild, Rake)

* + **Todos comitean sus cambios a la rama principal frecuentemente.**

Antes que nada hay que recordar como funcionan los repositorios: comitear significa subir nuestros cambios, existe una copia local y en el servidor de nuestro proyecto, nosotros trabajamos en local y cuando queremos guardar nuestros cambios primero tenemos que actualizar mi copia local con los cambios que se encuentran en el servidor xq posiblemente otro desarrollador ha subido previamente sus cambios, en ese momento mi copia se intentará integrar por si misma con los cambios del servidor y si por algún motivo no puede realizarlo me pedirá que yo resuelva unicamente los conflictos que ha encontrado (luego ejecuto mi build) Es decir cada vez que yo hago un commit estoy integrando mi trabajo con el de otros desarrolladores y también estoy resolviendo los errores que puedan haber.

Si recordamos al inicio de la charla, los problemas de integración se producían xq lo que teníamos que integrar era tanto que era muy difícil hacerlo y tomaba mucho tiempo. Entonces si nosotros subimos nuestros cambios constantemente significa que lo que estamos integrando es una porción pequeña pero funcionando de nuestro trabajo, si esta porción es muy pequeña las posibilidades de que existan errores en el build será muy pequeña y aún más pequeña la probabilidad de conflictos en el código; pero si por algún motivo se da el error, este error lo estamos descubriendo de manera oportuna y también será más fácil de resolver.

Esta práctica es la más importante de todas, podremos tener un buen servidor, un build script pero si subimos nuestros cambios constantemente no tenemos nada. Por eso es muy importante si es que vamos a introducir esta práctica que incentivemos a nuestro equipo a que suba sus cambios constantemente, varias veces al día es lo ideal, al menos 2 o 3 veces al día es lo justo y necesario.

* + **Se debe ejecutar el Build por cada commit en un máquina destina para integración.**

Una de las cosas que se quiere al realizar commits frecuentes es que la rama principal dentro del repositorio se mantenga siempre estable, sin errores de código ni tampoco funcionales. Pero cuando realizamos un commit existe la posibilidad de que muchas cosas puedan salir mál, desde que el desarrollador se olvidó actualizar primero desde el repositorio, no corrió el build localmente o se olvidó de subir algún archivo lo que ocasiona que el código dentro del repositorio no funcione o talvez existe diferencias en los ambientes.

Para evitar esto es que se recomienda ejecutar el build en una tercera máquina y solo si ahí pasa el build el commit se considera válido. Hay 2 formas de realizar:

* + - Manualmente
    - Automáticamente: Aca es donde entran los denominados Continuous Integration Servers. Estos CIS no unos monitores que están constantemente chequeando el repositorio, apenas detectan que alguien ha realizado un commit, descargan la utlima versión de repositorio, ejecutan el build automáticamente y notifican el resultado. ES casi como si contratáramos a una nueva persona que su único trabajo fuera mirar el repositorio y ejecutar el build.

Existen varias clases de servidores de CI: Hudson, Cruise Control, Teamcity.

Recuerden los pasos:

* + - * lo primero que tenemos que hacer es darle un nombre y una descripción, por lo tanto al inicio de todo CREAMOS EL PROYECTO
      * el CIS monitorea el repositorio, por lo tanto necesitamos CONFIGURAR REPOSITORIO
      * ejecuta Build, CONFIGURAR LOS PASOS DEL BUILD
      * cuando falla el build, CONDICIONES DE FALLA
      * este build se ejecuta cada vez que se hace commit, CONFIGURAR CUANDO SE LANZA EL BUILD

Creamos el proyecto y en la primera pantalla tiene muchas opciones, pero no nos debemos preocupar la clave para configurar el servidor de integración es leer cuidadosamente y dirigirnos a las opciones que nos permitan hacer cada uno de estos pasos. Asimismo a medida que vayamos configurando tenemos que ir probando.

* + - * Primero el repositorio
      * El Build y los build steps, aca agregamos cada paso dentro de nuestro build, agregamos la compilación y probamos, agregamos el test y probamos. Antes de seguir agregando más steps pasamos a la siguiente configuración.
      * Ahora los triggers y probamos.
  + **Keep the build Fast.**

Uno de los aspectos más importantes de la IC es poder recibir un feedback muy rápido, pero no hay nada peor que un build que demore demasiado. Por eso debemos tratar en lo posible de disminuir el tiempo del build ya que también ayuda a ahorrarle tiempo a los desarrolladores. Incluso el primer libro de XP recomienda que el build no debería pasar de 10 minutos, que en el peor de los casos es un tiempo en el cuál te puedes parar ir un momento al baño o comprarte un café.

Se imaginán que ejecutemos todas tareas en cada commit, demoraría demasiado. Si bien estas tareas deben pertenecer a un build xq son importantes no deben pertenecer a ESTE build.

Para estos casos hay varias estratégias:

* + - Pipeline build (builds en secuencia). El commit puede lanzar el primer build en el cuál se encuentran las tareas de las cuáles necesitamos un feedback muy rápido. Al finalizar este build puede notificar rápidamente si es que hubo un problema y sino desencadenar automáticamente un segundo build que pueda tener por ejemplo los tests más lentos o tareas contra la base de datos.
    - Ejecutar los builds en diferentes momentos de tiempo: seguramente mucho de nosotros hemos escuchado el término “daily build”, este término usualmente se usa cuando no se tratan de builds lanzados por acciones humanas, sino builds que son programados para lanzarse en un momento del tiempo. Por ejemplo en la noche para que los resultados estén al inicio del día, en el fin de semana, etc.

Estos 2 enfoques son totalmente compatibles podríamos tener un build en secuencia en el cuál el primer build: compile,test. El segundo build ejecute test contra la base de datos y verifique el cimplimiento de ciertas métricas. Y podríamos tener tener el build nocturno que haga todas estas tareas pero adicionalmente genere un instalador, generé documentación como un javadoc o realice un despliegue a algún ambiente.

Para decidir que es lo que va en cada build, necesitamos balancear que tantos errores pretendemos encontrar y la velocidad del build de tal manera que permita trabajar a todo el equipo.

(CREAR EL NIGHTLY BUILD)

* + **Todos pueden ver qué es lo que está pasando**

Uno de los aspectos más importantes de la IC es la visibilidad y la comunicación de tal manera que todos podamos ver cuál es el estado del sistema y de los cambios realizados en el.

Uno de los aspectos más importantes de está comunicación es conocer siempre el estado del build. Si no conocemos el estado del build nunca vamos a saber si existe un problema, no podremos arreglarlo rápidamente y a las personas no le importarían, y todo esto de la integración continua no tendría ningún sentido.

Para esto tenemos varias alternativas que nos brinda el mismo SIC, la primera es el website,emails, notificaciones directamente a nuestro escritorio etc etc.

Para ver esto vamos a probar que es lo que pasa cuando se produce un error en el servidor.

Build Failure and Notification (Email, Tray) Hacerlo fallar ver en el tray y luego arreglarlo. Si no tenemos una forma de notificación no hay forma de ver cuál es el estado del build.

* **Cuales son sus beneficios**
  + El código dentro del version control funciona a algún nivel.
  + Pasamos menos tiempo integrando ya que se integra en cada commit del código y no tenemos que esperar una gran cantidad de tiempo para saber que lo que hicimos funciona. Ya que es más fácil realizar la integración en partes pequeñas y mucho más fácil encontrar el error si es que se produce.
  + Incrementa la visibilidad ya que todos pueden dirigirse.
  + Elimina el riesgo no solo por xq integramos y sabemos que lo integrado funciona constantemente sino xq la visibilidad elimina al incertidumbre que tienen los stakeholders de lo que sucede con el proyecto. (eliminar errores tarde es más costoso, pobre calidad de código, software que no puede ser desplegado)
  + Elimina la fricción entre los miembros del proyecto, ya que los testers no tienen que acercarse al desarrollador para pedirle la una versión específica ni los líderes de equipo tienen que exigir a su equipo ciertos datos ya que pueden ir directamente al servidor, todo esto hace que el flujo de trabajo dentro del equipo fluya mucho mejor.
  + Menos tiempo creando y desplegando builds. La generación de artefactos y el despliegue suelen ser toda una ceremonia, pero cuando estos se realizan de forma automatizada, continua y con un simple click no solo pierdes el miedo que traen estas tareas sino ahorras gran cantidad de tiempo y evitas realizar múltiples errores.