Despliegue continuo

El despliegue continuo es una técnica permitiendo de aplicar nuevo código constantemente y de manera inmediata a un proyecto. Este articulo, escrito por Eric Ries, el cofundador de IMVU, un sitio de chat con más de 20 millones de usuarios, nos hace analizar las ventajas de esta práctica, y también destaca el hecho que se hace falta una cierta organización para implementarla. Pero esta organización también permite de aportar un rigor non despreciable al proyecto, para poder ser más eficiente y reducir los errores al máximo, avanzando por lo tanto mas rápidamente.

En su introducción el autor también destaca que el despliegue continuo es un sujeto controversial, ya que sin buena disciplina, esta técnica lleva a errores y a un código de mala calidad : de hecho si todos los creadores del proyecto "push" constantemente sin preocuparse del trabajo de los otros, por supuesto da confusiones y desordena todo el trabajo realizado hasta ya, y implica que siempre hay que volver atrás para corregir los errores recientemente creados. Eso es precisamente lo que queremos evitar, y por eso hay que respetar ciertas reglas antes de cada push. Estas últimas, según el autor, son fáciles de implementar, y su automatización permite de ganar mucho tiempo y rigor. Las presenta en cinco etapas.

Primero, lo básico para asegurarnos que no llevamos ningún error en el código es testar permanentemente lo que estamos escribiendo, que debe funcionar por supuesto con la parte sobre cual trabajamos pero también con el proyecto en su integralidad, por eso cada vez que agregamos algo al proyecto entero se debe asegurar que todo queda en orden con lo que se hizo antes. La elección de los testes es a propósito muy importante por eso, ya que a la vez hay que testar todo lo que podría andar mal pero no se debe olvidar que no podemos pasar demasiado tiempo en testes tampoco. Por eso el autor recomiende que el servidor pase menos de 30 minutos para efectuarlos todos, si no hay que repartirlos en diferentes computadores para quedar eficientes. También da un consejo para crear nuevos testes : cada vez que se arregla un error añadimos uno en esta parte del código, para reducir las posibilidades de error donde se hace mas falta, identificando las fuentes mas rápidamente. Así contribuimos a la robustez del proyecto mientras se construye. Con eso en cabeza nos propone softwares, como BuildBot que es lo cual usa, para automatizar estos testes cada vez que se hace un push, así reducimos de manera simple los riesgos que se agregue un error importante al proyecto final.

Su segundo punto es de permitir de detener el añadido de todo nuevo código cuando hay un problema. De hecho si el proyecto presenta un bug o cualquier cosa que no funciona como lo queríamos, agregar nuevo código podría llevar solamente a agregar nuevos errores encima de los otros , y eso sería mas y mas difícil de corregir volviendo atrás. Por eso es importante cada vez que se hace un push que el proyecto ya ande correctamente, si no hay que corregir primero lo que se hace falta. Así el sistema debe ser capaz de rechazar un nuevo commit, con un commit-check script, y cada vez que un test falle debe prohibir nuevo código de ser agregado al proyecto. Como lo dice el autor, es importante ser rápido para llevar un proyecto a cabo, pero hay que guardar en cabeza que ir demasiado rápido no sirve para nada si no producimos un código de alta calidad, ya que eso significa que en un momento tendremos que volver atrás y por lo tanto nos causa más retraso al final.

En su tercero punto el autor nos habla sobre el script de despliegue. Aunque en IMVU tienen uno muy sofisticado, nos recomienda de empezar con algo simple, ya que ellos mismos no empezaron así con algo muy complicado. El proceso de despliegue debe quedar simple y claro para todos, aun así con tiempo evolucionará y se automatizará cada vez mas. También la regla enunciada arriba se queda, eso es que no se puede cambiar el proyecto si todavía todos los testes no han sido realizados.

Pero aun así con estas tres reglas, se puede que un error ingrese al proyecto, escapando a los testes que no siempre cubren la totalidad del proyecto. También, incluso si escribimos muchos testes y de manera completa, siempre se puede fallar un detalle, y a veces toma horas, o días antes de que se nota. Por eso, un otro punto importante del sistema propuesto por el autor es de ser capaz de avisar en tiempo real a un miembro del equipo cada vez que se encuentra un bug así, para que pare la "línea de producción" como lo llama el autor, es decir impedir el sistema de aceptar nuevos commits de acuerdo con el commit-check script, y luego por supuesto para corregirlo. Con ese fin nos recomienda utilizar open source softwares como Nagios, que permiten de vigilar datos del sistema como el promedio de carga del servidor o el uso de memoria, lo que es muy útil para asegurarse de que no hay un error allí. Cuando detecta una anomalía, el sistema debe alertar al equipo, sonando un teléfono por ejemplo así que se puede resolver rápidamente. No debe sonar demasiado tampoco porque de esta manera no lo prestaríamos bastante atención, debe ser para problemas urgentes principalmente, entonces lo debemos configurar eligiendo pocos factores al menos al principio.

Una vez que se resolvió un problema de tipo urgente como aquellos de que acabamos de hablar, es importante discutirlo con el resto del equipo antes de empezar a trabajar de nuevo. Eso es el quinto y último punto importante presentado por el autor para implementar el sistema de despliegue continuo. Este proceso lo llama "Five Whys", porque para cada problema que puede ocurrir, consista en hacer cinco preguntas recursivamente para remontarse a la raíz de lo que sucedió. Estas preguntas se enfocan cada una en una capa diferente del sistema, por ejemplo "porque no tuvimos este problema antes con nuestro canal de despliegue", "porque no fuimos avisados por Nagios cuando el bug ocurrió", "porque nuestro proceso de despliegue dejo el problema entrar", etc. Es fundamental hacerlo porque resulta en mejorías en cada uno de los cinco niveles del sistema analizados, lo que poco a poco contribuye a la robusteza del proyecto.

Se puede que todas estas reglas solamente resuelvan pocos errores, pero serían los más difíciles de arreglar, y pues al final permiten de ganar mucho tiempo unas vez implementadas, así que merece la pena de tomar un poco de tiempo para ponerlas en marcha. Hay un principio famoso que se dice 80/20 : para cada proyecto, 80% del tiempo se dedica para 20% del código, entonces limitar al máximo estos pequeños problemas permite de ahorrar mucho tiempo y dolores de cabeza, por eso tomar una o dos semanas para instalar los softwares citados arriba, implementar las dichas reglas y luego tomar el tiempo de aplicarlas puede volverse muy beneficioso para el proyecto final.

Por fin, despliegue continuo es un proceso largo de implementar de manera eficiente. Sin embargo, demanda poco esfuerzos por lo que resulta, ya que con tiempo es una técnica presentada aquí con muy poderosa y alcanzable para todos. En este articulo, el autor no presentó sus defectos, que se manifestarían sobretodo en caso de mala organización, lo que podría rápidamente resultar en muchos errores de varios lados. Por eso es importante ser muy riguroso usando despliegue continuo, para sacarle mayor provecho.