Análisis y Diseño de Sistemas 2

Entrega continua (continuous delivery)

Material de apoyo del curso Análisis y Diseño de Sistemas 2 de la USAC

Ing. Ricardo Morales

Primer semestre 2016

Análisis y Diseño de Sistemas 2

Entrega continua (continuous delivery)

Tabla de contenido

OBJETIVOS

- Describir el concepto de entrega continua
- Conocer y entender los elementos que integran un sistema de entrega continua y su relación con un sistema de control integración continua
- Describir y entender el concepto de deployment pipeline
- Identificar el valor de las prácticas recomendadas para implementar la entrega continua

DEFINICIÓN

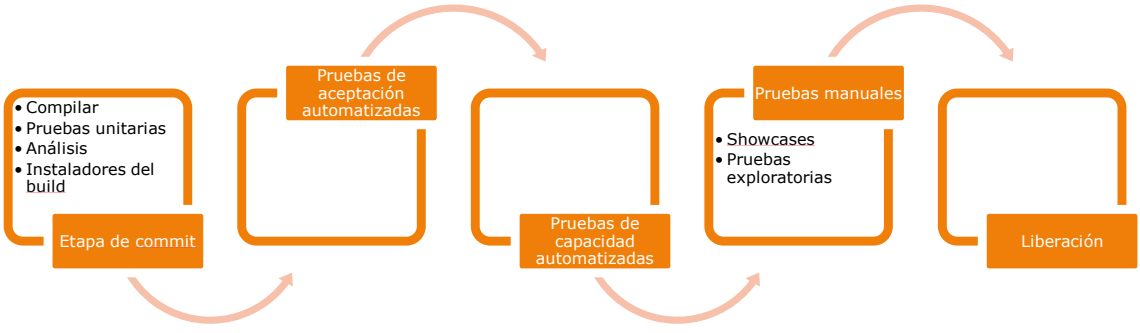
Es la extensión natural de la integración continua: un enfoque en el cual los equipos aseguran que daca cambio al sistema es publicable (releseable), y que podemos publicar cualquier versión al presionar un botón.

La entrega continua busca que hacer los releases sea aburrido, de manera que se pueda entregar frecuentemente y obtener retroalimentación rápida acerca de lo que les importa a los usuarios.

Si alguien piensa que algo es una buena idea, ¿cómo lo entregamos a los usuarios tan pronto como sea posible?

La entrega de software ocurre luego de que los requerimientos se han identificado, la aplicación se ha diseñado y desarrollado.

Para que las actividades de entrega sean confiables y coordinadas (entre testers, desarrolladores y operaciones), se propone el uso del *deployment pipeline.*



ANTI PATRONES

Publicar software manualmente

Los signos de este anti patrón son:

- La producción de documentación detallada y extensa que describe los pasos a seguir y la forma en que los pasos pueden salir mal
- Dependencia en pruebas manuales para confirmar que la aplicación esta corriendo correctamente
- Llamadas frecuentes al equipo de desarrollo para que explique porque el deployment tiene problemas el día de liberación
- Correcciones frecuentes al proceso de liberación, durante una liberación
- Ambientes en un cluster que difieren en su configuración
- Liberaciones que toman mas de pocos minutos para ejecutarse
- Liberaciones en las que no se puede predecir el resultado y que a veces debe darse marcha atrás

En vez de:

En el tiempo, los deployments deben tender a ser completamente automatizados. Debería haber 2 tareas ejecutadas por un humano para hacer un deployment en cualquier ambiente: elegir la versión y ambiente; y presionar el botón "deployar".

El liberar software empaquetado debe involucrar un proceso simple automatizados que crea el instalador

Publicar a un ambiente parecido a producción solo cuando se terminó el desarrollo

Este patrón se ve así:

- Si los probadores han sido involucrados hasta este punto, han probado el sistema en máquinas de desarrollo
- El publicar en este ambiente es la primera vez que operaciones interactúa con el nuevo release
- El ambiente es tan caro que el acceso está controlado estrictamente o no existe
- El equipo de desarrollo ensambla instaladores, archivos de configuración, migraciones de base de datos y documentación de deployment para pasarlo a la gente que en realidad hace el deployment, todo sin probarse en un ambiente que se parezca a producción
- Existe poca colaboración entre el equipo de desarrollo y la gente que ejecuta los deployments

En vez de:

Integrar las actividades de pruebas, deployment y release en el proceso de desarrollo. Se debe hacerlas como actividades normales y parte del desarrollo de manera que en el momento de liberar el sistema en producción haya poco riesgo, porque se ha liberado varias veces antes de eso.

Administración manual de la configuración del ambiente de producción

Signos de este patrón son:

- Luego de haber publicado varias veces en staging o preproducción, la publicación falla en producción

- Diferentes miembros de un cluster se comportan de forma diferente, por ejemplo sostener menos carga o tomar mas tiempo para ejecutar un proceso
- Al equipo de operaciones le toma mucho tiempo preparar un ambiente para una liberación
- No se puede regresar a una configuración previa del sistema, que puede incluir sistema operativo, servidores de aplicación, servidores web, base de datos u otras configuraciones de infraestructura
- Los servidores en clusters tienen, no intencionalmente, diferentes versiones de sistema operativo, librerías o niveles de parches
- La configuración del sistema se lleva a cabo modificando la configuración directamente en los sistemas de producción

En vez de:

Todos los aspectos de los ambientes de pruebas, preproducción y producción deben ser aplicados desde el control de versiones a través de un proceso automatizado.

OBJETIVOS DEL PROCESO DE LIBERACIÓN

Los objetivos a alcanzar en un proceso de liberación son:

- Velocidad, reducir el tiempo de ciclo, el tiempo que toma desde decidir un cambio hasta tenerlo disponible a los usuarios.
- Alta calidad, nuestro software debe ser acorde a su propósito

Para alcanzar estos objetivos, se necesita hacer releases (liberaciones) automatizadas y frecuentes

Automatizado.

Si el proceso de build, deploy, pruebas y liberación no está automatizado, no es repetible.

Cada vez que se hace es diferente, por los cambios en el software, la configuración del sistema o el ambiente. Si los pasos son manuales, existe mas probabilidad de error y no hay forma de revisar exactamente que se hizo.

Esto significa que no se tiene control sobre el proceso y ni sobre la calidad

Frecuente.

Si los releases son frecuentes, el delta entre releases será pequeño. Esto reduce significativamente el riesgo asociado con liberar y hace mas fácil regresar a una versión previa. Los releases frecuentes llevan a una retroalimentación mas rápida

PROCESO DE RETROALIMENTACIÓN

Involucra probar cada cambio de una manera completamente automatizada, en la medida de lo posible. Las pruebas varían dependiendo del sistema, pero usualmente incluyen:

- El proceso de crear el código ejecutable debe funcionar. Esto verifica que la sintaxis del código fuente es valida

- Las pruebas unitarias del software deben pasar. Esto verifica que el código de la aplicación se comporta como se espera
- El software debe llenar ciertos criterios de calidad como cobertura de pruebas y otras métricas específicas a la tecnología
- Las pruebas de aceptación del software deben pasar. Esto verifica que la aplicación esté conforme a los criterios de aceptación del negocio
- Las pruebas no funcionales deben pasar. Esto verifica que la aplicación se ejecute suficientemente bien en términos de capacidad, disponibilidad, seguridad y otros para alcanzar las necesidades del usuario
- El software debe ir a pruebas exploratorias y demostraciones al cliente. Esto se hace típicamente de forma manual en el ambiente de pruebas. En este paso el dueño del producto decida si hay características faltantes o errores que requieren cambios

La retroalimentación es esencial para releases frecuentes y automatizados. Hay 3 criterios para que la retroalimentación sea útil:

- Cualquier cambio, de cualquier tipo, necesita disparar el proceso de retroalimentación
- La retroalimentación debe obtenerse tan pronto como sea posible
- El equipo debe recibir la retroalimentación y actuar con base en ella

Cualquier cambio, de cualquier tipo, necesita disparar el proceso de retroalimentación

Los componentes que pueden cambiar y deben ser sujetos de control son:

- Código ejecutable.

  Cambia cuando se cambia el código fuente.

  Cada vez que se hace un cambio al código fuente, el binario resultante debe compilarse y probarse.

  Para tener control sobre el proceso, esto debe ser automatizado. A esta práctica se le llama integración continua.

  Este código ejecutable debe ser el mismo ejecutable que se despliega en cada ambiente.

- Configuración

  Cualquier cosa que cambie entre ambientes debe capturarse como información de configuración.

- Ambiente

  Si el ambiente en el que la aplicación se desplegará cambia, el sistema completo debe ser probado con los cambios del ambiente.

  Esto incluye cambios en configuración de sistema operativo, el software que soporta la aplicación, configuración de red y cualquier infraestructura.

- Datos

  Si la estructura de datos cambia, este cambio también debe probarse.

La retroalimentación debe obtenerse tan pronto como sea posible

La clave para la velocidad es la automatización. Con procesos automatizados, la única restricción es la cantidad de hardware con que se cuenta.

Las pruebas en la etapa de commit se caracterizan así:

- Corren rápido
- Son tan completas como sea posible, cubren mas 75% del código base
- Si alguna de ellas falla, significa que la aplicación tiene una falla crítica y no debe liberarse bajo ninguna circunstancia
- Son neutrales al ambiente como sea posible, lo que significa que pueden mas simples y baratas

Las pruebas en etapas posteriores tienen las siguientes características generales:

- Corren mas lentamente y por lo tanto son candidatas para paralelización
- Algunas pueden fallar y podemos elegir liberar la aplicación bajo algunas circunstancias
- Deben correr en un ambiente tan parecido como sea posible al de producción, también prueban el proceso de deployment y cualquier cambio al ambiente de producción

El equipo debe recibir la retroalimentación y actuar con base en ella

Es esencial que todos los involucrados en el proceso de entrega de software estén involucrados en el proceso de retroalimentación. Esto incluye a desarrolladores, probadores, staff de operaciones, administradores de base de datos, especialistas de infraestructura y administradores.

Estar listo para reaccionar a la retroalimentación también significa distribuir información. Esto implica el uso de mecanismos que aseguren que la retroalimentación llegue a los integrantes del equipo.

Finalmente, la retroalimentación no es buena a menos que se actúe con base en ella. Esto requiere disciplina y planificación.

BENEFICIOS

El beneficio principal de este enfoque es que crea u proceso de liberación que es repetible, confiable y predecible, que genera reducción en el tiempo de cada ciclo y por lo tanto proporciona el software al usuario mas rápidamente

Empoderamiento de equipos

La habilidad de desplegar fácilmente cualquier versión del software en cualquier ambiente tiene varias ventajas:

- Los probadores pueden seleccionar versiones antiguas de una aplicación para verificar los cambios de comportamiento en las nuevas versiones
- El staff de soporte puede desplegar una versión liberada de una aplicación en un ambiente para reproducir un defecto
- El staff de operaciones puede seleccionar un build correcto conocido para desplegar en producción como parte de un ejercicio de recuperación de desastres
- Las liberaciones pueden ejecutarse con presionar un botón

Reducir errores

Las cosas que deben estar bien para que una aplicación funciones son la versión correcta del código fuente, del esquema de base de datos, la configuración correcta de balanceadores, el URL del web service, etc.

El manejo activo de todo lo que puede cambiar en un control de versiones permite que el proceso se automatizado y se reduzcan errores.

Otros beneficios son:

- Baja de stress
- Flexibilidad de despliegue
- La práctica hace la perfección

RELEASE CANDIDATO

¿Qué es un release candidato? Un cambio al código puede o no ser liberado.

Si se analiza el cambio y se pregunta ¿deberíamos liberar este cambio?, la respuesta sería adivinar.

Es el proceso de build, deployment y pruebas que aplicamos a ese cambio, el que valida si el cambio puede ser liberado.

Este proceso incrementa nuestra confianza en que el cambio es seguro de liberar, al hacer verificaciones automatizadas en el menor tiempo posible.

Este enfoque es diferente al que identifica release candidatos al final del proceso, usando versiones alpha, beta, etc.

PRINCIPIOS DE ENTREGA CONTINUA

Crear un proceso repetible y confiable para liberar software

Liberar software debe ser fácil porque se ha probado cada parte del proceso de liberación cientos de veces.

Desplegar software involucra 3 cosas:

- Proveer y administrar el ambiente en el cual correrá la aplicación (configuración hardware, software, infraestructura y servicios externos)

- Instalar la versión correcta de la aplicación en el ambiente
- Configurar la aplicación, incluyendo datos o estado que se requiera

Automatizar casi todo

Hay algunas cosas que es imposible automatizar, como pruebas exploratorias, demostraciones a usuarios o aprobaciones de cumplimiento. Sin embargo, la lista de coas que no se pueden automatizar es mas pequeña de lo que usualmente se cree.

Muchos equipos de desarrollo no automatizan su proceso de liberación porque parece una tarea de enormes proporciones. Es mas fácil hacer las cosas manualmente.

La automatización es el prerequisito para el deployment pipeline, porque es solo a través de la automatización que se obtendrá lo que se necesita. Sin embargo, no se necesita automatizar todo de una vez, se puede iniciar con las partes del proceso que son un cuello de botella.

Mantener todo en control de versiones

Todo lo que se necesita para hacer bulid, deploy, pruebas y release de la aplicación debe ser mantenido en alguna forma de almacenamiento versionado, identificando esto con un identificador.

Debe ser posible para un nuevo integrante del equipo sentarse en su máquina, obtener la versión del proyecto del repositorio y correr un comando simple para hacer el build y desplegar la aplicación en cualquier ambiente, incluyendo su máquina.

Si duele, hacerlo mas frecuentemente

Si liberar software es doloroso, busquemos liberar cada vez que alguien hace un cambio que pasa todas las pruebas automatizadas.

Dependiendo del nivel de expertise, esto podría llevar mucho tiempo. Se puede buscar metas intermedias, como generar un release interno cada semana. Gradualmente se podría llegar al ideal.

Construir la calidad

Lo mas pronto que se identifiquen defectos, será mas barato corregirlos.

Las pruebas no son una fase y ciertamente no una fase que se inicie al final de la fase de desarrollo.

Las pruebas no son de dominio exclusivo de los probadores, todos en el equipo son responsables de la calidad de la aplicación todo el tiempo.

Hecho (finalizado) significa liberado

Una característica está finalizada solo cuando da valor a los usuarios. Esta es la motivación tras la práctica de integración continua.

Para algunos equipos "finalizado" significa liberado en producción, esto puede no ser práctico siempre, la mejor opción que sigue es que la funcionalidad haya sido demostrada a los usuarios en un ambiente parecido a producción.

Un corolario de esto es que no depende de una persona tener algo finalizado, requiere que el equipo trabaje en conjunto para eso.

Todos son responsables del proceso de entrega

Idealmente, todos en una organización están alineados con sus objetivos y la gente trabaja en conjunto para alcanzarlos.

En varios casos, el desarrollador "tira" su trabajo sobre la pared a los desarrolladores, quienes hacen lo mismo hacia operaciones. Cuando existe un problema, se gasta tiempo culpándose entre sí.

El tener un sistema donde todos pueden ver el estado de la aplicación, que pruebas ha pasado y el estado del ambiente donde se publica, facilita la comunicación.

Mejora continua

El proceso de entrega debe ser evaluado periódicamente para buscar la mejora del mismo.

DEPLOYMENT PIPELINE

CI asegura que el código que se crea como equipo funcione, proveyéndonos retroalimentación rápida de problemas introducidos con un commit, se enfoca en equipos de desarrollo.

La salida de un sistema de CI generalmente va a pruebas o deploys manuales, lo que se puede ver como desperdicio, ya que puede tomar mucho tiempo llevarlo a un ambiente de producción.

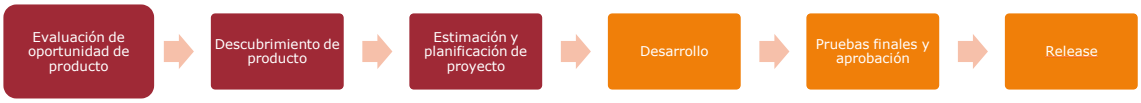
La solución es adoptar un enfoque mas holístico para entregar software.

Lo que se tiene es un sistema de donde se extrae (pull system). Los equipos de pruebas y operaciones pueden obtener un build y desplegarlo en el ambiente que necesiten, con solo presionar un botón.

¿Qué es un deployment pipeline?

A un nivel abstracto, un deployment pipeline (tubería de instaladores) es una manifestación automatizada de su proceso para obtener software desde el control de versiones hasta las manos de sus usuarios.

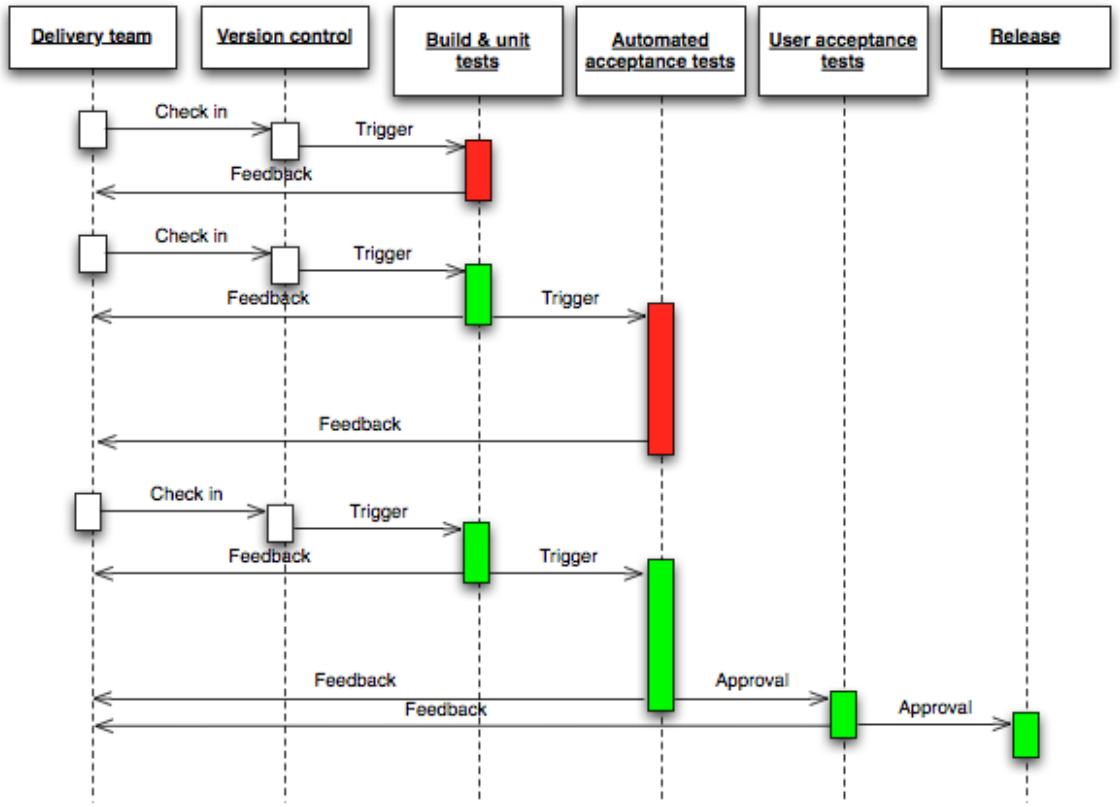
En la siguiente gráfica se puede ver la cadena de valor para un producto. Entre cada paso, transcurre un tiempo que agrega valor y un tiempo para pasar de un paso a otro (de espera, que no agrega valor).



La parte de la cadena que se busca mejorar es la que va de deployment hasta release.

Existen consecuencias importantes de aplicar este patrón:

- Se está previniendo efectivamente la liberación (release) en producción de builds que no han sido completamente probados y no cumplen con su propósito
- Cuando los deployments y liberaciones a producción están automatizados, son rápidos, repetibles y confiables



Fases de un deployment pipeline

- Etapa de commit

  Verifica que el sistema funciones al nivel técnico. Compila, pasa un conjunto de pruebas automatizadas y corre análisis de código

- Etapas de pruebas de aceptación automatizadas

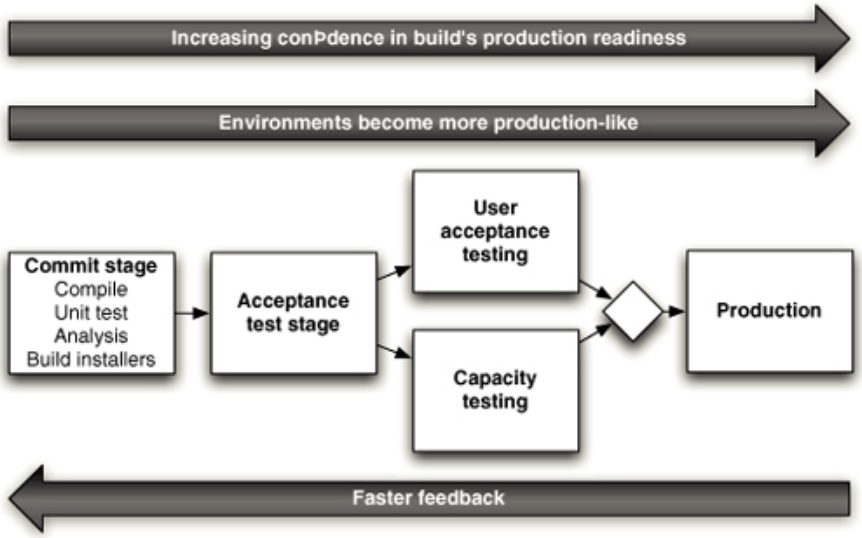
  Verifica que el sistema funciona al nivel funcional y no funcional

- Etapas de pruebas manuales

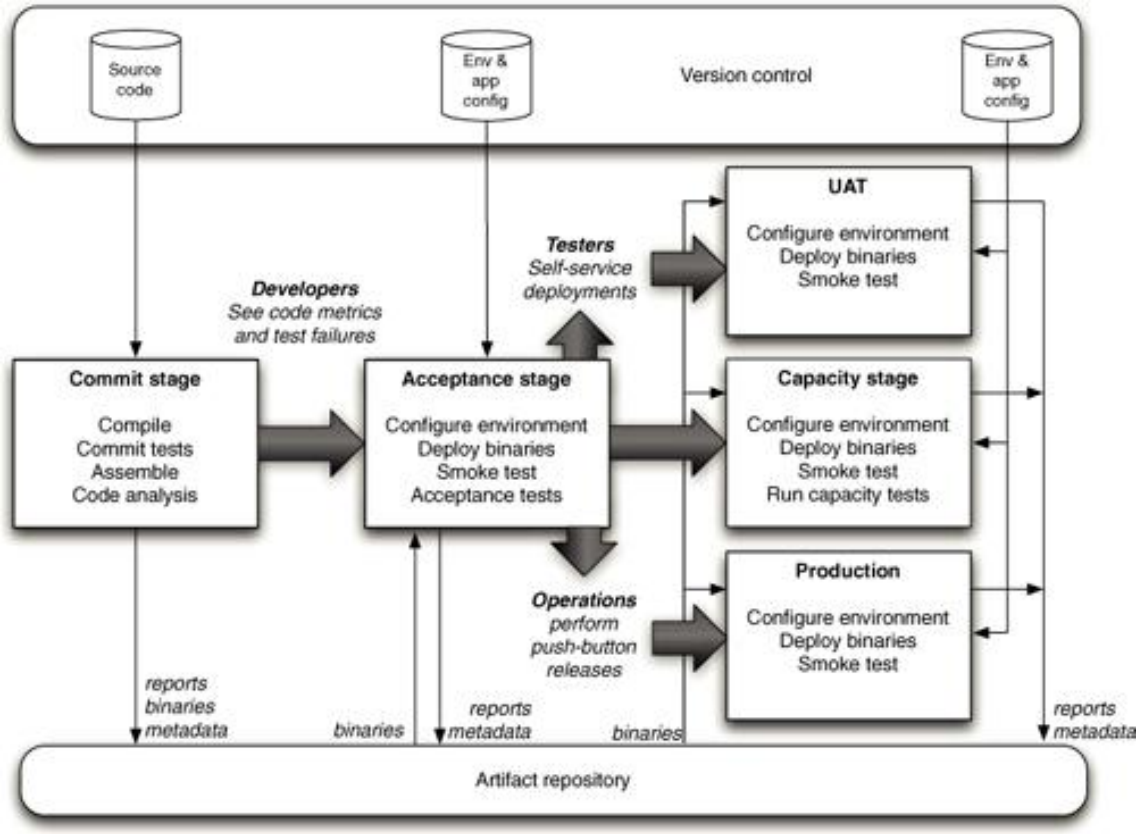
Verifica que el sistema es usable y llena sus requerimientos, detecta defectos no encontrados antes y verifica que provee valor a los usuarios

- Etapa de release

  Lleva el sistema a los usuarios



Deployment pipeline básico



El proceso inicia con los desarrolladores haciendo commit de sus cambios en el sistema de control de versiones. El sistema de CI dispara una nueva instancia del pipeline.

La primera etapa (commit) compila el código, corre pruebas unitarias, ejecuta análisis del código y crea instaladores. Si todo funciona bien, se crean binarios y se almacenan en un repositorio de artefactos.

La segunda etapa generalmente está compuesta por pruebas de aceptación automatizadas que tardan mas. Esta etapa se dispara con la ejecución exitosa de la etapa previa.

En este punto, el pipeline se divide para permitir un deployment independiente en varios ambientes, en este caso: UAT (User Acceptance Testing), pruebas de capacidad y producción.

Finalmente, es importante recordar que el propósito de todo esto es obtener retroalimentación tan rápido como sea posible. Para hacer rápido el ciclo de retroalimentación, se necesita poder ver que build se despliega en que ambiente y en que etapas del pipeline ha pasado cada build.

Prácticas deployment pipeline

Solo hacer build de los binarios una vez

Se referirá como binarios a los ejecutables, según la plataforma.

Usualmente se compilan los binarios cada vez que se publicará en una ambiente diferente. Esto viola 2 principios importantes.

- El primero es mantener el deployment pipeline eficiente, para obtener retroalimentación tan pronto como sea posible. Recompilar cada vez viola este principio
- El segundo es siempre mantener el build bajo fundamentos conocidos, los binarios que llegan a producción deben ser los mismos que pasaron las pruebas

Al recrear los binarios, se corre el riesgo que se introduzca algún cambio entre la creación de los mismos y su release, de manera que solo se debería hacer el build de binarios durante la etapa de commit.

Desplegar de la misma manera en cada ambiente

Es esencial usar el mismo proceso para desplegar en cada ambiente para asegurar que el proceso de build y despliegue sea probado efectivamente.

La frecuencia de despliegue es inversa al riesgo asociado con cada ambiente.

Cada ambiente es diferente de alguna manera, esto no significa usar un script de deployment diferente, sino mantener los settings diferentes separados y aplicarlos apropiadamente según el ambiente.

Si se usa el mismo proceso para desplegar en todos los ambientes, cuando algo no funcione en un ambiente, se puede reducir a 3 causas:

- Un setting en un archivo de configuración de la aplicación especifico del ambiente
- Un problema asociado con la infraestructura o uno de los servicios de los cuales depende la aplicación
- La configuración del ambiente

Hacer pruebas de humo de los deployments

Pruebas de humo son pruebas no exhaustivas que buscan asegurar que las funciones mas cruciales de una aplicación funcionen, sin preocuparse de detalles mas finos.

Puede ser tan simple como ejecutar la aplicación y obtener la pantalla de ingreso para verificar que sea correcta.

También debe verificar que cualquier servicio del que dependa la aplicación este arriba y corriendo (bd, bus, servicios externos).

La prueba de humo o prueba de deployment es probablemente la prueba mas importante a escribir, después de las pruebas unitarias, ya que da la confianza que la aplicación realmente corre.

Desplegar en una copia de producción

El otro problema principal que experimentan los equipos es que los ambientes de pruebas o desarrollo son bastante diferentes del ambiente de producción.

En la medida del presupuesto, se deberían tener copias exactas de producción, lo cual requiere asegurarse de:

- La infraestructura, como topología de red y configuración de firewall es la misma
- La configuración del SO es la misma, incluyendo parches
- El stack de aplicaciones es el mismo
- Los datos de la aplicación están en u estado conocido y valido

Cada cambio debe propagarse a través del pipeline instantáneamente

La primera etapa debe ser disparada luego de cada commit y cada etapa debe disparar a la siguiente inmediatamente, luego de una ejecución exitosa.

Por el tiempo que pueden tomar las etapas, es posible que esto no sea posible.

Si al terminar una etapa, no es posible ejecutar la siguiente porque aún está corriendo para el build anterior, se puede utilizar una planificación inteligente.

Una planificación inteligente hará que se corran las etapas que correspondan, solo para el commit mas reciente.

Si cualquier parte del pipeline falla, detener la línea

Si un deployment a un ambiente falla, el todo el equipo es dueño del fallo. Deben detenerse y corregirlo antes de hacer otra cosa

Descripción de etapas

La etapa de commit

Esta etapa incluye los siguientes pasos

- Compilar el código (si es necesario)
- Correr un conjunto de pruebas de commit. Incluyen las pruebas unitarias y pueden incluir pruebas de aceptación mas importantes

- Crear binarios para uso en etapas posteriores
- Ejecutar análisis del código para verificar su salud. En el análisis se pueden establecer métricas como: cobertura de pruebas, cantidad de código duplicado, complejidad ciclomatica, acoplamiento aferente y eferente, número de warnings y estilo del código
- Preparar artefactos, tales como bases de datos de prueba, para uso en etapas posteriores

Etapa de pruebas de aceptación automatizadas

Sin correr pruebas de aceptación en un ambiente parecido a producción, no sabemos nada acerca de si la aplicación satisface las especificaciones de los clientes o si puede ser desplegada y sobrevivir en el mundo real.

La mayoría de pruebas que corren en esta etapa son de aceptación, pero pueden haber mas.

La meta de la etapa de pruebas de aceptación es asegurar que el sistema proporciona el valor que el cliente está esperando y que satisface los criterios de aceptación.

También sirve como una prueba de regresión, verificando que no se hayan introducido errores por nuevos cambios.

El equipo debe responder inmediatamente a un problema en las pruebas de aceptación. Deben decidir si el problema es resultado de la regresión, un cambio en el comportamiento de la aplicación o un problema con la prueba, tomando la acción apropiada.

Etapas subsecuentes de pruebas

La etapa de pruebas de aceptación es un hito significativo en el ciclo de vida de un release candidato, ya que el release se mueve de algo que es mayormente el dominio del equipo de desarrollo a algo de mayor interés y uso.

El paso de una etapa a la siguiente ha sido automático, en esta etapa se puede elegir entre continuar automáticamente con la siguiente etapa o hacerlo de manera manual.

Por ejemplo, se puede hacer un deployment a un ambiente de pruebas para hacer pruebas manuales.

Acá es importante el apoyo de una herramienta que provea la habilidad de ver que está desplegado en cada ambiente y hacer un despliegue a través de un botón a cada ambiente.

Con lo anterior, los probadores pueden elegir un release candidato, desplegarlo a un ambiente de pruebas y hacerle pruebas manuales. La idea es verificar los criterios de aceptación, mas que hacer pruebas de regresión.

El ejecutar pruebas no funcionales como un requisito para hacer una publicación en producción, dependerá de cada equipo y situación. Aún en el caso de que se ejecuten, el llevar una aplicación a producción podría decidirse con base en la revisión de los resultados obtenidos y no de forma automática.

Etapa de release

Entre menos control se tenga sobre el ambiente en el cual se ejecuta el código, hay mas potencial que para comportamientos no esperados, de manera que al queremos estar en control cada cosa que se despliegue.

Hay 2 factores que se oponen a este ideal

- No se tiene el control del ambiente operacional del software

  Por ejemplo juegos o aplicaciones de oficina. El problema se mitiga seleccionando muestras representativas de los ambientes destino y corriendo las pruebas automatizadas en dichos ambientes

- Se asume que el costo de implementar ese grado de control es mayor que los beneficios

  Usualmente es cierto lo contrario: la mayoría de problemas en producción son causados por control insuficiente

El proceso de despliegue puede automatizarse a través de scripts de build y deployment.

Con un proceso automatizado de deployment y release, el proceso de entrega se democratiza.

Hay 2 razones por las que se teme a los días de release:

- El miedo de introducir un problema por un error difícil de detectar de una persona al realizar los pasos manuales de release o porque hay un error en las instrucciones
- Si el release falla, por el proceso o un defecto en la versión del software, se está comprometido

El primer problema se mitiga al ensayar el release muchas veces al día, probando que el sistema automatizado de deployment funcione.

El segundo es mitigado a través de una estrategia de "back-out" (marcha atrás).

Para el momento que un release candidato está disponible para despliegue en producción, sabemos que las siguientes afirmaciones son verdaderas:

- El código compila
- El código hace lo que los desarrolladores piensan que debe hacer porque ha pasado sus pruebas unitarias
- El sistema hace lo que los analistas o usuarios piensan que debe hacer porque ha pasado las pruebas de aceptación
- La configuración de infraestructura y ambiente base es administrada apropiadamente, porque la aplicación ha sido probada en un análogo de producción
- El código tiene todos los componentes correctos porque fue desplegable
- El sistema de deployment funciona porque, como mínimo, ha sido usado en el release candidato al menos una vez en el ambiente de desarrollo, una en la

etapa de pruebas de aceptación y una en el ambiente de pruebas, antes que el release se promueva a esta etapa
- El sistema de control de versiones tiene todo lo que se necesita para desplegar, sin necesidad de intervención manual, porque se ha desplegado el sistema varias veces

Implementando el deployment pipeline

- Modelar su cadena de valor y crear un esqueleto que funcione

  Consiste en identificar el proceso actual y generar un mapa de la cadena de valor, para modelarlo en las herramientas de integración continua y administración de release

- Automatizar el proceso de build y deployment

  El proceso de build (que genera los "binarios") debe ejecutarse cada vez que se hace commit, esto se puede hacer con las herramientas de integración continua

  El siguiente paso es automatizar el deployment, que puede incluir crear paquetes de la aplicación e instalarla y configurarla. Inicialmente el deploy puede ser en el servidor de CI, pero luego debe mejorarse a otros ambientes

- Automatizar pruebas unitarias y análisis de código

  Significa correr pruebas unitarias, análisis de código y una selección de pruebas de aceptación e integración en cada commit

- Automatizar pruebas de aceptación

  Debe buscarse incluir pruebas funcionales y no funcionales, al menos alguna de ellas desde el inicio del proyecto

- Evolucionar el pipeline

  El pipeline puede evolucionar a pipelines por componentes o uso de branches, es importante recordar:

  - No se necesita implementar el pipeline de una sola vez
  - El pipleline es una fuente rica de datos acerca de la eficiencia del proceso
  - El pipeline es un sistema vivo

Métricas

Para el proceso de entrega de software, la métrica global mas importante es tiempo de ciclo, que es el tiempo desde que se decide que una característica debe ser implementada hasta que se libera a los usuarios.

Otros diagnósticos que pueden alertar acerca de problemas son:

- Cobertura de pruebas automatizadas
- Propiedades del código base (duplicación de código, complejidad ciclomatica, etc.)
- Número de defectos
- Número de commits por día

- Número de builds por día
- Número de fallas de builds por día
- Duración del build, incluyendo pruebas automatizadas

RESUMEN GRÁFICO

Un resumen gráfico de entrega continua puede encontrarse en: http://continuousdelivery.com/2014/02/visualizations-of-continuous-delivery/