Attlasian Bamboo

Victor Herrero Cazurro

Bamboo

Contenidos

1. ¿Que es Bamboo?

Bamboo es un servidor de integración continua (CI).

Se puede emplear para automatizar la verificacion del estado de salud de un proyecto de software, asi como para crear un pipeline de entrega continua (continuous delivery).

La herramienta ofrece numerosos plugins para extender las tareas basicas soportadas.

- Para la obtención de los fuentes del software ofrece compatibilidad con numerosas herramientas como Bitbucket Server (Stash), Bitbicket Cloud, GIT, GitHub, SVN, CVS, . . .
- Para la construcción igualmente con herramientas como MSBuild de Microsoft, Maven y gradle para el mundo java, . . .
- Una vez construido el software, permite realizar otras tareas con los artefactos generados, como crear ZIP, MSI, despliegue en servidores, ejecucion de scripts, . . .

Y todo ello desde una inerface Web.

2. ¿Que aporta Bamboo?

- Posibilidad de automatizar el proceso de compilación y de pruebas, haciendolo mas confiable
- Posibilidad de contruir de distintas formas un mismo proyecto, según el escenario.
- Despliegue automatico en plataformas como App Store o Google Play.
- Definición de un entrono independendiente y aislado.
- Posibilidad de lanzar el proceso automaticamente al presentarse una nueva version del software.
- Pemite optimizar el rendimiento de la compilación del software al lanzar procesos paralelos.
- Permite deplegar de forma automatica (continuous delivery) por ejemplo para permitir realzar pruebas de usuario final (prueba de aceptación del usuario (UAT)).

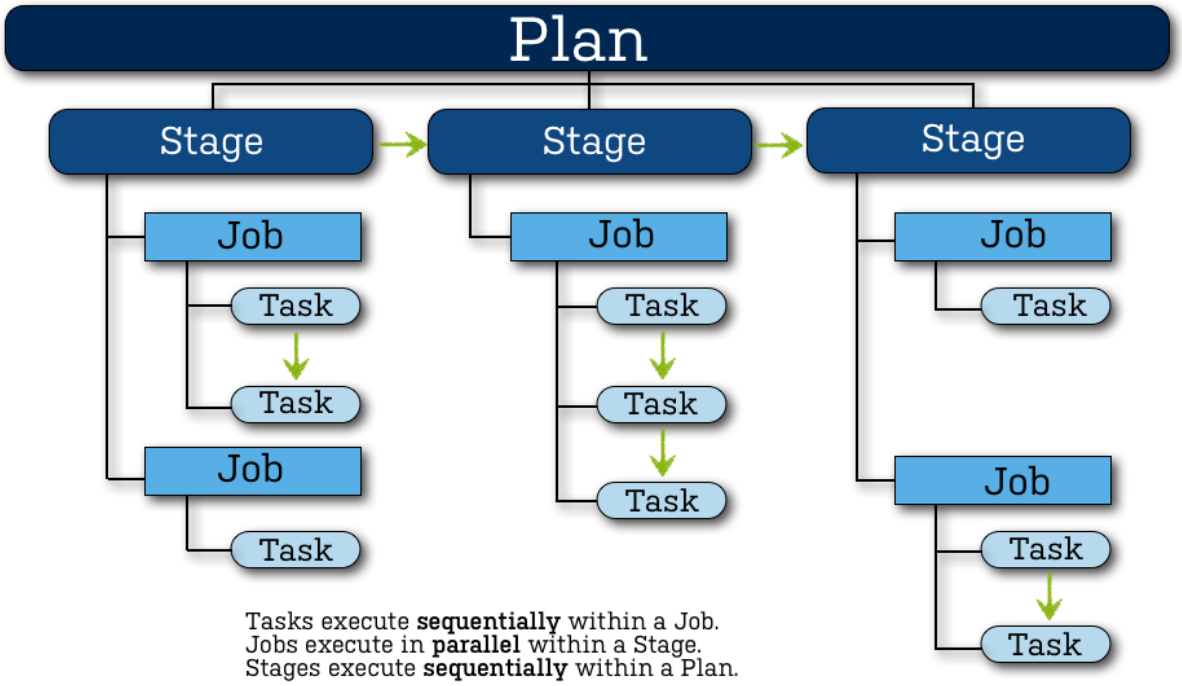
3. ¿Que necesita Bamboo para definir un proceso?

- El origen de los fuentes del Software
- Los scripts a aplicar a dichos fuentes para obtener los artefactos.
- Conjunto de pruebas que validan el correcto funcionamiento del artefacto generado.

4. Flujo de trabajo

El flujo de trabajo de Bamboo define los siguientes actores

- Proyecto
- Plan
- Etapa
- Trabajo
- Tarea



4.1. Proyecto (Project)

- Elemento principal, que permite agrupar planes de construcción relacionados.

- Puede tener asociado mas de un plan.
- Proporciona informes para cada Plan a traves del panel de control.
- Proporciona enlaces a otras aplicaciones.
- Permite configurar permisos para todos los planes que contiene.

4.2. Plan

- Representa la construcción de un proyecto de software.
- Por defecto define una unica Etapa, pero se pueden definir más para agrupar trabajos.
- Las Etapas se procesan de forma secuencial utilizando el mismo repositorio.
- Especifica el repositorio predeterminado.
- Especifica cómo se desencadena la compilación y las dependencias desencadenantes entre el plan y otros planes en el proyecto.
- Especifica notificaciones de resultados de compilación.
- Especifica quién tiene permiso para ver y configurar el plan y sus trabajos.
- Proporciona la definición de las variables del plan.
- Los proyectos y planes solo pueden ser configurados por los administradores de Bamboo.

4.3. Etapa (Stage)

- Represena cada una de las etapas por las que ha de pasar el software para poder ser construido. Por ejemplo.
  - ¬ Compilacion, ejecución de Test Unitarios y paquetizacion.
  - ¬ Despliegue en QA.
  - ¬ Ejecución de Test de Integracion en QA.
  - ¬ Despliegue en Producción (Será manual)
  - ¬ Ejecucion de Test de Humo en Producción.
- Por defecto define una unico Trabajo, pero se pueden definir más para que se procesen en paralelo y aprovechar los agentes..
- Se deben completar con éxito todos los trabajos de una Etapa antes de que se pueda procesar la siguiente Etapa del plan.

- Puede producir artefactos que pueden estar disponibles para su uso en una Etapa posterior.
- Las etapas solo pueden ser configuradas por los administradores de Bamboo.

4.3.1. Etapas manuales

- Cualquier etapa de un plan puede ser una etapa manual.
- Hacen que Bamboo pause la ejecución del plan cuando se llega a ellas, continuando unicamente cuando el usuario las active manualmente
- Solo se puede activar si la etapa anterior se completó con éxito.
- No se pueden omitir.
- Se muestran en el Navegador de planes con un icono distinto.
- Se necesita el permiso "Build" en el plan para ejecutarlas.

4.4. Trabajo (Job)

- Define un conjunto de tareas que se procesan en paralelo, en múltiples agentes (allí donde estén disponibles), un ejemplo de uso sería la ejecucion de distintos escenarios de pruebas.
- Se descompone en una serie Tareas.
- Controla el orden en que se realizan las tareas.
- Reúne los requisitos de tareas individuales en el trabajo, de modo que estos requisitos se puedan combinar con las capacidades del agente.
- Define los artefactos que generará la compilación.
- Solo puede usar artefactos producidos en una etapa previa.
- Especifica cualquier etiqueta con la que se etiquetarán el resultado de compilación o los artefactos de compilación.

4.5. Tarea (Task)

- Es una pequeña unidad de trabajo discreta, como la comprobación del código fuente, la ejecución de un Goal Maven, la ejecución de un script o el análisis de los resultados de las pruebas.
- Se ejecuta secuencialmente dentro de un trabajo en un directorio de trabajo de Bamboo.

5. Instalacion

Se puede obtener el producto de aquí en distintos formatos de instalación Zip, Windows, Linux o MAC OS X.

Además existen imagenes en Dockerhub con la herramienta instalada como esta.

Para ejecutar el servidor, se precisa de una licencia, para evaluación se provee una de 30 dias de forma gratuita si se dispone de una cuenta en Atlassian. Se pueden consultar las licencias obtenidas aquí. == ¿Que es Integracion Continua (Continuous Integration)?

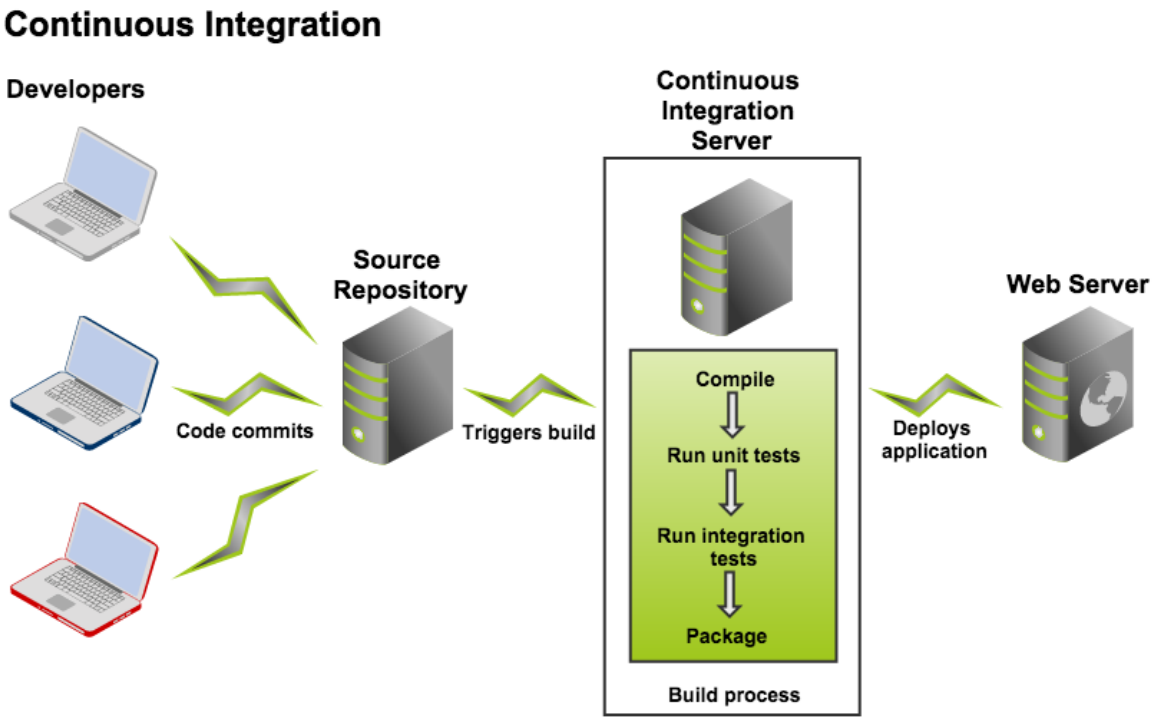
Metodología de desarrollo de software que permite evaluar el estado de un proyecto en un momento determinado, determinando si los últimos cambios introducidos en el software han introducido además problemas que hagan que el software este inestale.

El proceso normalmente es automatico, desencadenandose su ejecución de forma planificada o bien por aparecer una nueva version del software.

Las versiones del software se extraerán de un sistema gestor de versiones.

La evaluación del estado del proyecto se basará en pruebas de compilación, pruebas unitarias y pruebas de integración.

El objetivo principal de estos sistemas es la detección temprana de problemas en los nuevos desarrollos.



5.1. Ventajas de IC

- Llegan menos errores a producción por el descubrimiento temprano.
- No existen problemas en crear versiones del software, ya que al hacerse periodicamente, los problemas ya han sido detectados.
- Los desarrolladores sufren menos cambios de contexto, ya que al verificarse el estado del proyecto nada mas subir sus cambios, si existen errores, son avisados antes incluso de que hayan cambiado a otra tarea.
- Los costos temporales por la ejecución de las pruebas se reducen drásticamente: el servidor de su CI puede ejecutar cientos de pruebas en cuestión de segundos y de forma automatica.
- El equipo de calidad se puede dedicar a otras tareas, como definicion de procedimientos, o profundizar en las pruebas ya que no han de dedicar su tiempo con las pruebas que sean automatizadas.

5.2. Costes de IC

- Se precisan pruebas automatizadas para cada nueva característica, mejora o corrección de errores.
- Servidor de integración continua y repositorio de versiones de software.
- Commit muy frecuentes al repositorio de versiones de software, recomendable al

menos una vez al día.

6. ¿Que es Entrega Continua (Continuous Delivery)?

Consiste en automatizar la generación del software entregable.

El concepto esta asociado a las metodologias agiles de desarrollo, por lo que en principio esta asociado a iteraciones breves de desarrollo que provocan software funcional.

Evidentemente no se puede llegar a automatizar la entrega de un producto si este no tiene la calidad suficiente, por lo que todo lo comentado en referecia a la importancia de las pruebas para la IC es igualmente aplicable a la EC, si las pruebas son confíables, suponen una garantía de calidad, y por tanto es posible publicar el software cuando las pasan, si estas pruebas se pasan con gran frecuencia, se puede llegar a un escenario en el que en cualquier momento se puede entrega el software actualizado.

La entrega continua no siempre significa entregar, sino tener la posibilidad de hacerlo rapidamente.

6.1. Ventajas de EC

- Se elimina la complejidad de la generación del entregable, por lo que no hay que dedicarle dias a la preparación.
- Al poder entregar mas rapido, se obtiene un feedback antes.
- Los cambios peqeños so mas facilmente implementables, no existe la sensación de riesgo por el cambio.

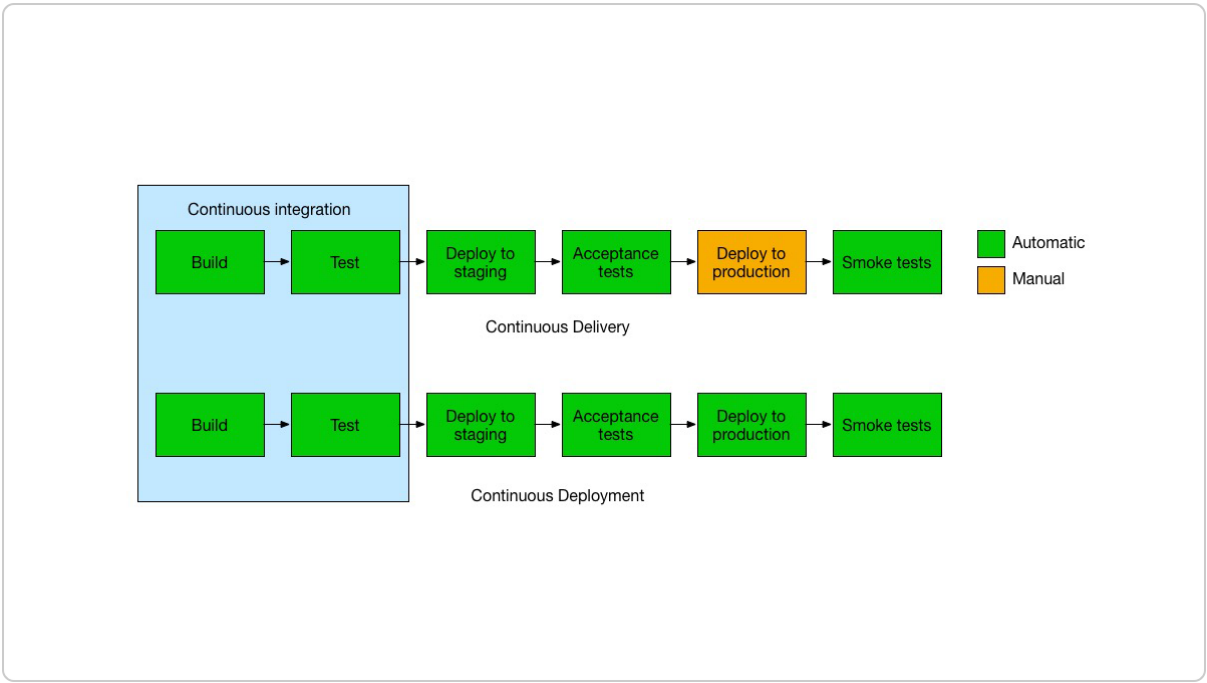
6.2. Costes de EC

- Se necesita bunas bases de integración continua, con test que cubran gran cantidad de código, todo lo que no se cubra, se incluira en el entregable sin validar.
- Los despliegues deerías ser automatisados, con interacción humana para que se produzca, pero formado por un proceso automatico.
- Se deberán emplear perfiles para que funcionalidades que no esten completas no sean accesibles por el cliente final, aunque el codigo desarrollado ya este en el despliegue.

7. ¿Que es despliegue Continuo (Continuous Deployment)?

Es un paso más aplicado al concepto de Entrega Continua, en el que si se llega a desplegar el software en producción, sin ningun tipo de intervención humana, la unica forma de evitar una nueva release es que fallen los test.

Es una forma rapida de tener la última versión del software en funcionamiento y así poder tener el feedback del usuario final.



7.1. Ventajas de DC

- Puede desarrollarse más rápido ya que no es necesario detener el desarrollo para generar una release. Los despliegues se lanzan automaticamente con cada cambio.
- Las Releases son menos arriesgadas y más fáciles de parchear en caso de problemas.
- Los clientes ven un flujo continuo de mejoras y la calidad aumenta todos los días, en lugar de cada mes, trimestre o año.

7.2. Costes de DC

- de nuevo los test serán vitales, sin unos buenos test, se desplegará un software de poca calidad.

- La documentación se debe mantener al día.
- El uso de perfiles que permitan activar funcionalidades será obligatorio, dado que le software llega si o si a producción.

8. Buenas practicas para Integracion Continua

- Escribir test para las partes mas criticas de la aplicación.
- Programar un servicio de IC para que ejecute las pruebas cada vez que haya un cambio en el repositorio de fuentes
- Asegurate de que los desarrolladores integran sus cambios todos los dias, para que se pueda verificar el estado del proyecto diariamente (muy frecuentemente).
- Arregla los problemas tan pronto se detecten, el objetivo es que el estado del proyecto sea estable.
- Escribir Test para cada nuevo codigo generado, incuidos los arreglos (Fix).

9. Entornos de depliegue

- **Local**: Equipo del desarrollador
- **Desarrollo**: Servidor donde se realizan los test unitarios
- **Integration**: Servidor donde se realizan los test de integracion.
- **QA**: Servidor donde un perfil cliente realiza pruebas completas de la aplicación siguiendo un plan de pruebas.
- **Stage**: Servidor clon de Production.
- **Production**: Servidor donde esta la Aplicacion en uso por los clientes finales.