Introducción al desarrollo basado en Tronco

Definición e historia:

Antes de definir el TBD tengo que definir lo que es un branching model:

Definición de Branching Model:

Un Branching Model se refiere a una estrategia o enfoque estructurado para gestionar las distintas líneas de desarrollo de un proyecto de software. En esencia, se trata de la creación y gestión de ramas (branches) dentro de un repositorio de control de versiones, como Git.

Cada rama puede representar una característica, una corrección de error o una versión específica del proyecto, y se utiliza para aislar y controlar los cambios relacionados con esa tarea o función en particular. La gestión de ramas se convierte en una parte esencial de la colaboración en el desarrollo de software, ya que permite a los equipos trabajar en paralelo sin interferir entre sí.

El TBD es un branching model que se usaría junto con un sistema de control de versiones como git. En este modelo los desarrolladores colaboran en el código en una sola rama llamada “trunk” o tronco, sería la rama que conocemos como main. En este modelo los desarrolladores se alejan de la idea de crear ramas de desarrollo de larga duración y optan por el uso de ramas más pequeñas.

Actualmente algunas grandes empresas como Google o Facebook implementan este modelo a escala.

Principios básicos de TBD

El TBD es un modelo muy amplio, pero podemos extraer algunos principios básicos

1. **Colaboración en una sola rama (trunk/main).**
2. **Resistencia a las ramas de desarrollo de larga duración.**
3. **Mantenimiento de la integridad del release.**

Beneficios generales del modelo:

1. **Agilidad y Velocidad de Desarrollo.**
2. **Colaboración Efectiva.**
3. **Reducir la Complejidad.**
4. **Mejora Continua.**

Integración y entrega continuas

La Integración Continua (CI) y la Entrega Continua (CD) son prácticas esenciales en el desarrollo de software moderno, especialmente en el contexto de TBD. En este modelo, el rol de TBD es fundamental para habilitar un flujo de trabajo ágil donde los cambios al código se integran y despliegan de manera continua y eficiente.

Rol de TBD en la facilitación de CI/CD

TBD promueve la integración y prueba constante de cambios en el código en una rama principal o "trunk". Esta aproximación reduce significativamente los conflictos de integración y facilita un ciclo de desarrollo más fluido. Al trabajar de esta manera, se asegura que el código en el "trunk" esté siempre en un estado listo para ser desplegado, lo cual es un requisito fundamental para la CD.

Prácticas clave y técnicas

Para implementar eficazmente el TBD, existen varias prácticas y técnicas que los equipos deben considerar.

Ramas de características de corta duración vs. commit directo al trunk: La elección entre utilizar ramas de características de corta duración o hacer commits directos al trunk depende de la estrategia de gestión de cambios del equipo. Las ramas de características permiten un desarrollo aislado de nuevas funcionalidades sin afectar la estabilidad del trunk, pero deben ser fusionadas rápidamente para evitar divergencias significativas. Por otro lado, los commits directos fomentan una integración continua, pero requieren una disciplina y coordinación elevadas para mantener la calidad y estabilidad del código.

Uso de flags de características/microservicios y abstracción de ramas: Los flags de características son una técnica poderosa para manejar la introducción de nuevas funcionalidades sin modificar el funcionamiento del sistema. Permiten activar o desactivar funcionalidades en tiempo de ejecución, facilitando pruebas y despliegues graduales. La abstracción de ramas, por su parte, refiere a técnicas que minimizan la complejidad operativa de trabajar con múltiples ramas, simplificando la gestión del código.

Importancia de la revisión de código continua y CI para mantener la calidad: Las revisiones de código y la integración continua son fundamentales para asegurar la calidad y la coherencia del software desarrollado bajo el modelo de TBD. Las revisiones permiten identificar errores, mejorar la calidad del código y compartir conocimientos dentro del equipo.

La CI, por su parte, garantiza que cada cambio pase por un conjunto estándar de pruebas automatizadas, asegurando que el código en el trunk sea siempre de alta calidad y esté listo para ser desplegado.

Estos aspectos, cuando se implementan, pueden mejorar significativamente la eficiencia y calidad del desarrollo de software, permitiendo a los equipos responder de manera más ágil a las necesidades del proyecto y del mercado.

Gitflow vs TBD. Enfoques técnicos

El desarrollo basado en troncos (TBD) y GitFlow representan dos enfoques fundamentales en la gestión del desarrollo de software, adoptados ampliamente por compañías líderes como Google, Amazon, Netflix y Facebook. Mientras que TBD promueve la integración continua al hacer que todos los desarrolladores contribuyan al menos un commit diario directamente a la rama principal, facilitando las revisiones de código rápidas y la implementación de cambios menores, GitFlow ofrece una estructura más rígida, con ramas específicas para características, lanzamientos y desarrollos, lo cual permite un control más detallado sobre las versiones y facilita la gestión de proyectos a largo plazo o de gran escala. Este último se recomienda especialmente para equipos grandes, donde se minimiza la interferencia entre colaboradores mediante el trabajo en ramas separadas.

Ambos enfoques enfrentan desafíos como la necesidad de pruebas automatizadas y un equipo versátil y dispuesto a aprender nuevas tecnologías para garantizar la eficacia del desarrollo. Mientras que el desarrollo en tronco se beneficia de un ambiente de colaboración estrecha, aumentando potencialmente la productividad y la cohesión del equipo, también puede generar conflictos si no se maneja correctamente la superposición de código. La transición de GitFlow a TBD debe ser gradual, requiriendo el apoyo no solo de la dirección sino de todas las partes involucradas para asegurar una migración exitosa del modelo de desarrollo.