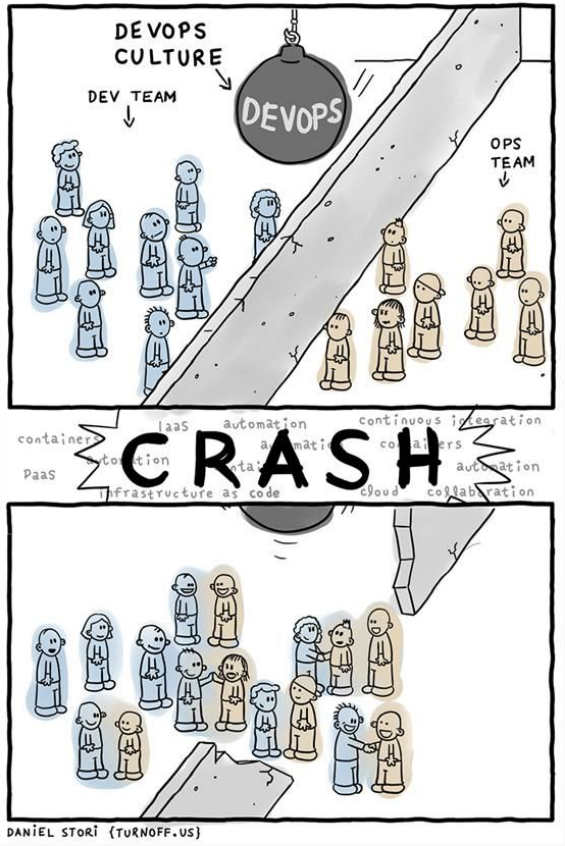
Unidad 4:
Automatización y seguridad en el despliegue

Módulo Puesta en Producción Segura

Integración del desarrollo y operación (DevOps)

- **DevOps (Developments & Operations)** es una combinación de desarrollos y Operaciones que podríamos interpretarlo como **Técnicas de desarrollo ágil.**
- DevOps permite que los roles que antes estaban aislados (desarrollo, operaciones de TI, ingeniería de la calidad y seguridad) se coordinen y colaboren para producir productos mejores y más confiables. Al adoptar una cultura de DevOps junto con prácticas y herramientas de DevOps, los equipos adquieren la capacidad de responder mejor a las necesidades de los clientes, aumentar la confianza en las aplicaciones que crean y alcanzar los objetivos empresariales en menos tiempo.



Integración del desarrollo y operación (DevOps)

- Es una metodología que es un conjunto de procesos, métodos y colectiva del sistema, que se utiliza para promover el desarrollo de aplicaciones, operación y mantenimiento de aplicaciones y control de calidad (QA) departamento. La comunicación, la colaboración y la integración se utilizan con el objetivo de romper las barreras y las diferencias entre el desarrollo y las operaciones tradicionales.
- DevOps es una cultura, el ejercicio o la práctica de potenciar a los desarrolladores de software (DEV) y de operación y mantenimiento de equipos técnicos (OPS), que permite la construcción, pruebas, software de liberación a través de la entrega de software automatizado y cambios arquitectónicos. Es más rápido, frecuente y fiable, concretamente se trata de mejorar la comunicación y colaboración durante la administración e implementación de software, cuyo objetivo es ser más rápido, más fiable y la elaboración de productos de mayor calidad.
- DevOps no un es un tipo de software particular, sino un conjunto de procesos y métodos.

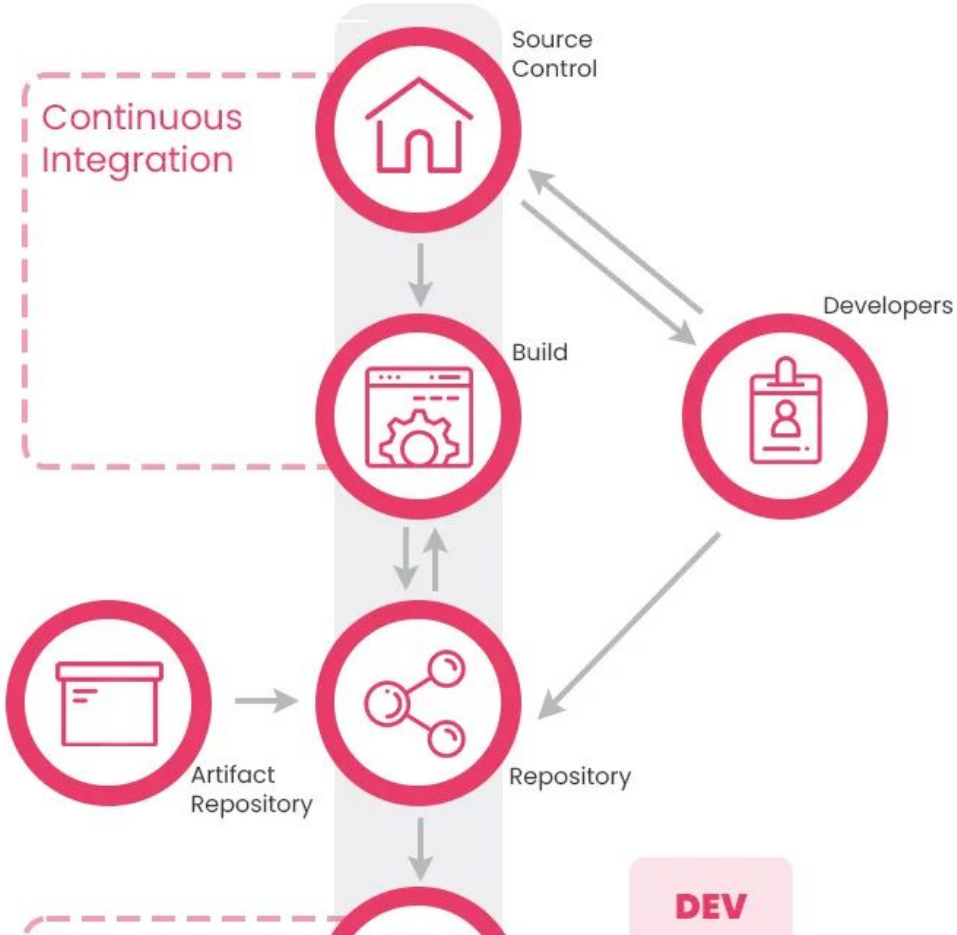
Figura 3.1: Estructura de los roles tradicionales frente a DevOps

DevOps y el ciclo de vida de las aplicaciones

- DevOps influye en el ciclo de vida de las aplicaciones a lo largo de las fases de planeamiento, desarrollo, entrega y uso. Cada fase depende de las demás y las fases no son específicas de un rol. En una auténtica cultura de DevOps, todos los roles están implicados de algún modo en todas las fases.
- Las fases que tenemos son: Planeamiento, Desarrollo, Entrega y funcionamiento.

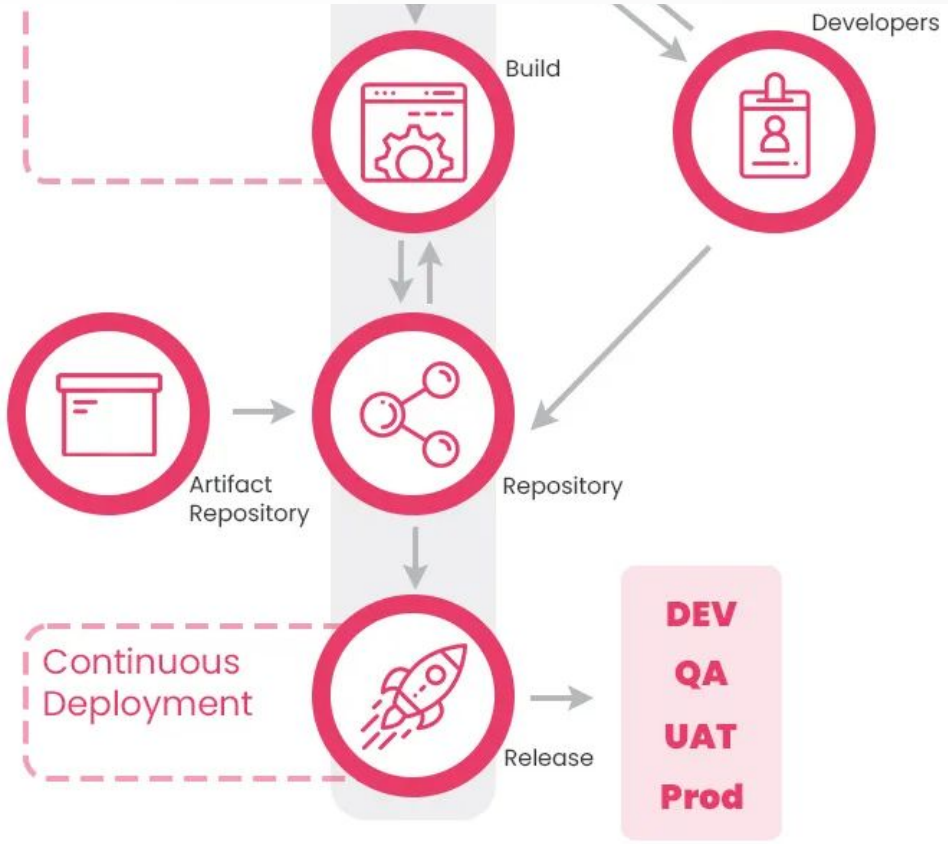
Integración del desarrollo y operación

- La **integración continua** (CI), la **entrega continua** (CD) y las **pruebas continuas (**CT) son los tres principales métodos mediante los cuales muchos equipos de desarrollo adoptan una estrategia ágil. Si bien cada uno cumple un objetivo ligeramente diferente, cuando se combinan, pueden ayudar significativamente al equipo a alcanzar la velocidad y la calidad, dos de los objetivos más importantes de cualquier equipo de desarrollo.
- Las **ventajas** que aporta son: Reducir la complejidad de proyectos largos, Reducir los errores, Facilitar el despliegue, Entregar más rápido y Mejorar los costes y la productividad



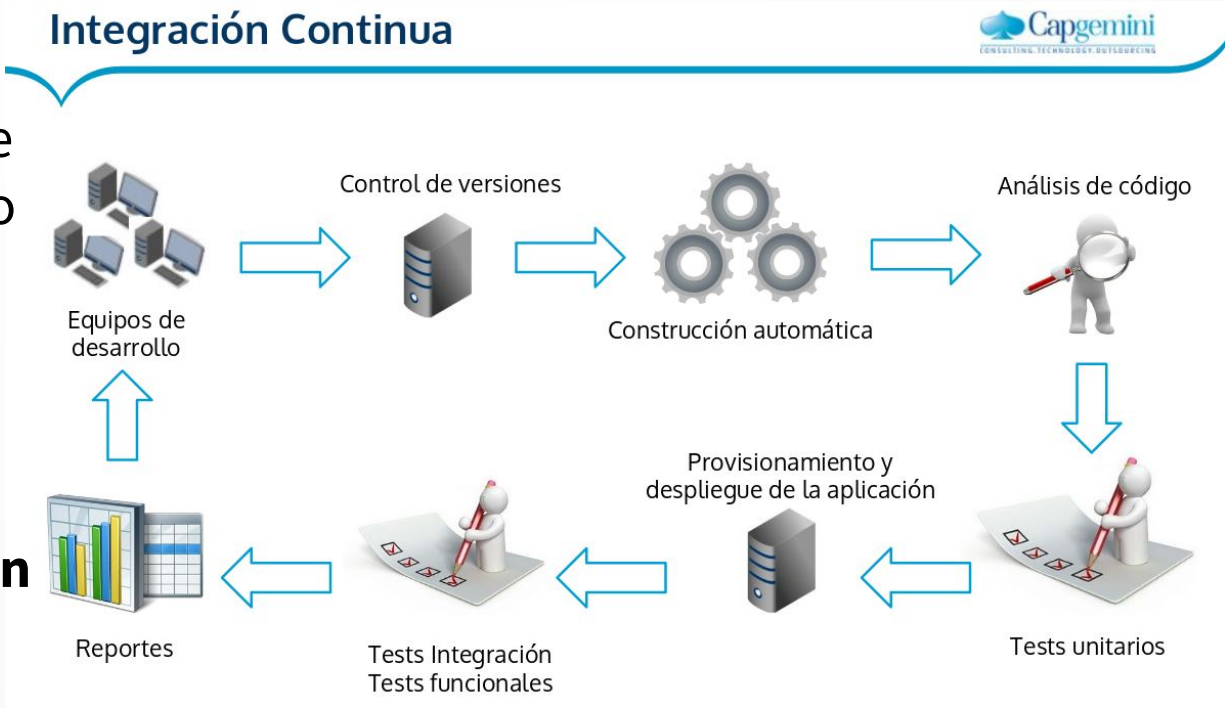
Integración del desarrollo y operación

- La **Integración Continua** («Continuous integration«) se centra en la integración del trabajo de los desarrolladores individuales en un repositorio principal varias veces al día para detectar errores de integración de manera temprana y acelerar el desarrollo colaborativo.
- La **Distribución Continua** («Continuous delivery «) tiene que ver con la reducción de la fricción en el proceso de implementación o liberación, la automatización de los pasos necesarios para implementar una compilación para que el código se pueda liberar de forma segura en cualquier momento.



Integración del desarrollo y operación

- La **Implementación Continua** («Continuous deployment«) va incluso un paso más allá al implementarse automáticamente cada vez que se realiza un cambio de código.
- Aparte de ellos también tenemos otros métodos cómo son CT (**Calidad Continua**) o IC **herramientas de automatización y procesos** de DC



Conceptos y prácticas clave para los Procesos Continuos.

- **Cambios pequeños e iterativos**: Los desarrolladores deben practicar el dividir el trabajo más grande en pequeños pedazos y combinarlos al repositorio apenas estén listos , y luego continuamente a lo largo del desarrollo. De esta forma el costo de la integración disminuye.
- **Desarrollo basado en tronco**: l trabajo se realiza en la rama principal del repositorio o se fusiona en el repositorio compartido a intervalos frecuentes. Las ramas con característica de vida corta son permisibles siempre que representen pequeños cambios y se fusionen lo antes posible.
- **Mantener rápidas las fases de construcción y prueba**: Cada uno de los procesos se basa en construcciones (compilaciones, N. del T.) y pruebas automatizadas para validar la corrección. Debido a que los pasos de compilación y prueba deben realizarse con frecuencia, es esencial que estos procesos se simplifiquen para minimizar el tiempo dedicado a estos pasos.

Control de versiones

- **Control de versiones** (o VCS, por sus siglas en inglés) es una práctica de software que consiste en administrar el código por versiones, haciendo un seguimiento de las revisiones y del historial de cambios para facilitar la revisión y la recuperación del código. Esta práctica suele implementarse con sistemas de control de versiones, como Git, que permite que varios desarrolladores colaboren para crear código.
- Estos sistemas proporcionan un proceso claro para fusionar mediante combinación los cambios en el código que tienen lugar en los mismos archivos y directorios, controlar los conflictos y revertir los cambios a estados anteriores.
- Un repositorio es un término del VCS que describe cuando un VCS está monitorizando un sistema de archivos. En el alcance los archivos individuales de códigos fuente, un VCS monitorizará las adiciones, eliminaciones y modificaciones de las líneas de texto que contiene ese archivo o directorio.
- El uso del control de versiones es una práctica de DevOps fundamental que ayuda a los equipos de desarrollo a trabajar juntos, dividir las tareas de programación entre los miembros del equipo y almacenar todo el código para poder recuperarlo fácilmente si fuese necesario.

Control de versiones

- Lo primero es crearse una copia local en nuestro dispositivo local.

Para aportar modificaciones hay dos esquemas o **formas de colaborar**:

- De **forma exclusiva:** en este esquema para poder realizar un cambio es necesario comunicar al repositorio el elemento que se desea modificar y el sistema se encargará de impedir que otro usuario pueda modificar dicho elemento. Una vez hecha la modificación, esta se comparte con el resto de colaboradores. Si se ha terminado de modificar un elemento entonces se libera ese elemento para que otros lo puedan modificar. P.e. SourceSafe, SubVersión lo incorpora.
- De **forma colaborativa**: en este esquema cada usuario modifica la copia local y cuando el usuario decide compartir los cambios el sistema automáticamente intenta combinar las diversas modificaciones. El principal problema es la posible aparición de conflictos que deban ser solucionados manualmente o las posibles inconsistencias que surjan al modificar el mismo fichero por varias personas no coordinadas. Además, esta semántica no es apropiada para ficheros binarios.P.e. subversion o Git permiten implementar este modo de funcionamiento.

Control de versiones

Podemos clasificar los sistemas de control de versiones atendiendo a la arquitectura utilizada para el almacenamiento del código:

- **Distribuidos**: cada usuario tiene su propio repositorio. Los distintos repositorios pueden intercambiar y mezclar revisiones entre ellos. Es frecuente el uso de un repositorio, que está normalmente disponible, que sirve de punto de sincronización de los distintos repositorios locales. Ejemplos: Git y Mercurial.
- **Centralizados**: existe un repositorio centralizado de todo el código, del cual es responsable un único usuario (o conjunto de ellos). Se facilitan las tareas administrativas a cambio de reducir flexibilidad, pues todas las decisiones fuertes (como crear una nueva rama) necesitan la aprobación del responsable. Algunos ejemplos son CVS, Subversion o Team Foundation Server.

Herramientas de control de versiones

Las principales herramientas de control de versiones son : CVS, Subversion, Git, Mercurial, Perforce, Team Fundation Server, Clearcase, etc..

Herramientas de control de versiones

https://www.monografias.com/trabajos99/gestion-configuracion-del-so
ftware/gestion-configuracion-del-software.shtml

- Ejemplo concreto Git (alternativa-> BitBucket):
  - https://aulasoftwarelibre.github.io/taller-de-git/cvs/
  - https://www.atlassian.com/git/tutorials/atlassian-git-cheatsheet
  - https://www.udemy.com/course/aprende-a-dominar-git-de-cero-a-expe rto/

Herramientas de control de versiones

Actividad:

Sigue el siguiente tutorial para aprender qué es Git y cómo se trabaja. Realiza las operaciones que se realizan en ella:

https://victorponz.github.io/Ciberseguridad-PePS/tema1/git/2020/11/01/Git.html

Puedes ayudarte también del siguiente enlace:

https://fp.josedomingo.org/iawgs/u01/github.html

1. Abre una cuenta en Github si no la tienes.
2. Crea un repositorio con nombre PPS4-1.
3. Sube a ella un programa de Hallo World en java.
4. Si es necesario crea el token para poder trabajar desde el cli.

Entrega un documento explicando estos 4 últimos pasos, no es necesario que adjuntes capturas de pantalla del proceso de instalación.