Tema 11

Introducción a DevOps

Ingeniería del Software Avanzada

Índice

[Esquema 3](#_Toc534735049)

[Ideas clave 4](#_Toc534735050)

[11.1. Introducción y objetivos 4](#_Toc534735051)

[11.2. Concepto de DevOps 5](#_Toc534735052)

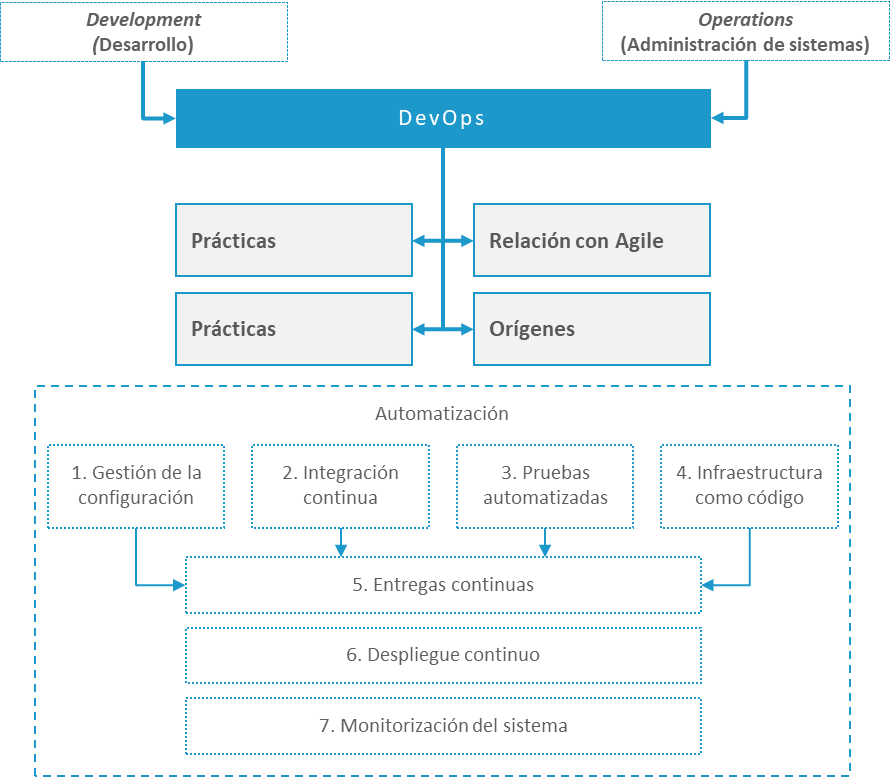
[11.3. Prácticas principales 12](#_Toc534735053)

[11.4. Referencias bibliográficas 28](#_Toc534735054)

[A fondo 32](#_Toc534735055)

[Test 35](#_Toc534735056)

Esquema



Ideas clave

11.1. Introducción y objetivos

**DevOps** es una cultura general que busca acercar o unificar en el seno de una compañía las **actividades de desarrollo de software —*development* (Dev)—** y las operaciones relacionadas con la **administración de sistemas —*operations* (Ops)—.** Mientras que los **equipos de desarrollo se centran en la construcción del software**, los **administradores de sistemas se encargan de gestionar el hardware** (servidores, infraestructura de red, etc.) y garantizar que se dan las condiciones adecuadas para que el equipo de desarrollo trabaje, se desplieguen las aplicaciones y el software funcione de manera óptima.

Ambos «mundos» han estado tradicionalmente separados en las empresas de desarrollo de software, y sus funciones han sido desempeñadas por perfiles a menudo muy diferentes. En los últimos años se ha visto la necesidad de acercar ambos grupos debido a **dos factores fundamentales**:

1. La **adopción de metodologías ágiles** —con iteraciones cortas y liberaciones continuadas de software— hacen necesario mejorar la colaboración entre los desarrolladores y equipo de operaciones para que el proceso transcurra de la manera más fluida posible.
2. La **popularización de las aplicaciones cloud**, casi ubicuas, y el hecho de que estas aplicaciones se desplieguen a menudo en infraestructura de terceros, que es consumida como un servicio más, configurable a través de las API, hace que las operaciones y el desarrollo sean actividades cada vez más próximas.

En este contexto surge **DevOps como un enfoque que enfatiza la monitorización de todo el proceso de construcción de software**, automatizando numerosas actividades relacionadas con la integración, pruebas y despliegue de las aplicaciones.

Con este tema alcanzaremos los siguientes objetivos:

* Comprender el concepto de **DevOps como cultura de desarrollo integrada**.
* Saber **relacionar** **DevOps con los conceptos de agilidad**.
* Conocer el **origen histórico de DevOps** como movimiento cultural.
* Conocer las **prácticas fundamentales** que permiten implementar DevOps.

11.2. Concepto de DevOps

Podemos definir DevOps como **«un conjunto de prácticas que persiguen la reducción del tiempo entre la introducción de un cambio en el código del sistema y la puesta en producción de ese cambio, al mismo tiempo que se garantiza la calidad del sistema»** (Bass, Weber & Zhu, 2015).

Otra definición que nos ofrecen Ott, Pham y Saker (2017) es que se trata de **«la cultura, principios y procesos que automatizan y agilizan el flujo de trabajo, de extremo a extremo, desde el desarrollo de código hasta la entrega de las nuevas características y cambios a los usuarios en producción».**

**Se trata de buscar la intersección entre desarrollo y operaciones, teniendo en cuenta la calidad del producto y del proceso en su conjunto**, como vemos en la figura 1. No se trata de una metodología, ni de un conjunto de prácticas estándar (aunque veremos algunas recomendadas), sino de **un enfoque cultural que debe ser interpretado y adaptado por cada organización a sus propias necesidades**.

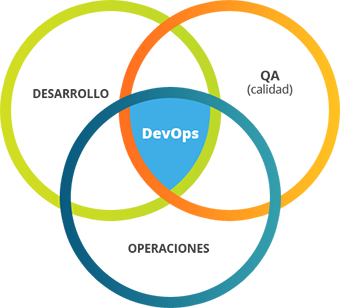


Figura 1. Representación esquemática del concepto DevOps. Fuente: Bello (2017)

Vemos también que esta idea está muy ligada a otras que aparecen en el origen de las metodologías ágiles. Encontramos palabras como «prácticas», «reducción de tiempo», «calidad», «cultura», «automatización»… Todas ellas vinculadas a conceptos de agilidad que hemos estudiado en temas anteriores. No en vano, **DevOps mejora el desarrollo ágil cubriendo una necesidad básica: la de tender un puente entre los equipos de desarrollo y operaciones, integrándolos en un «proceso y equipo unificados»** (Ott et al., 2017, p. 1). Esta idea aparece representada gráficamente en la figura 2. Al igual que sucede con otras metodologías desarrollo, Agile finaliza cuando comienza el despliegue a producción, y esto supone abrir una distancia entre los usuarios, desarrolladores y el equipo de operaciones.

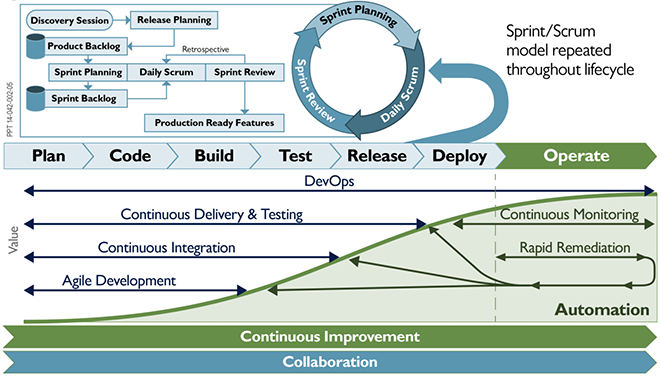


Figura 2. Relación entre DevOps y las metodologías ágiles, tomando Scrum como ejemplo.  
Fuente: Salmanson (2015)

**El término «DevOps» está íntimamente relacionado con el de «entregas continuas».** Como hemos visto, en el contexto tecnológico actual, cada vez es más importante proporcionar **actualizaciones frecuentes** a nuestros usuarios aumentando o mejorando funcionalidades. Las metodologías ágiles llevan esta idea al extremo, contemplando en principio la liberación de software funcional tras cada una de las iteraciones, a menudo de muy corta duración (una semana, por ejemplo).

El **concepto de la entrega continua implica que muchos procesos** —como las pruebas, el despliegue, o el propio desarrollo— **deben automatizarse** en la medida de lo posible y, por tanto, requiere una cierta adaptación por parte de la organización en su conjunto. En este sentido, la adopción de metodologías ágiles ha supuesto un enorme cambio en la manera en que las organizaciones desarrollan software. No solamente estas metodologías han surgido como respuesta a las necesidades cambiantes del entorno, sino que su adopción ha supuesto a su vez cambios en la manera en que estos productos se desarrollan y entregan a los clientes.

Desarrollo y operaciones

DevOps hace referencia al hecho de que **la práctica de la entrega continua se puede llevar al extremo**. Más que un concepto tecnológico, podría ser **un concepto cultural que abarca a la organización en su conjunto.**

Etimológicamente, el término proviene de las palabras ***Development***(desarrollo o conjunto de actividades para la construcción de un producto software) y ***Operations***(operaciones o conjunto de actividades, que realizan los administradores de sistemas para garantizar que están en condiciones óptimas para desplegar y ejecutar el software). Formalmente podemos definir estos conceptos del siguiente modo (Lwakatare, 2017):

1. **Desarrollo**: proceso que permite la creación de un producto software formado por un conjunto de actividades interrelacionadas como:
   * Análisis de requisitos.
   * Diseño arquitectónico.
   * Codificación.
   * Pruebas.
   * Integración.
   * Pruebas de aceptación.
2. **Operaciones**: proceso que pone en funcionamiento el sistema y ofrece al usuario final el soporte, durante su uso, en un entorno operacional. Aquí encontramos actividades como:
   * Instalación
   * Actualización
   * Migración
   * Monitorización
   * Gestión de configuración.
   * Alertas.
   * Disponibilidad.
   * Soporte.

Como vemos en la figura 3, **DevOps supone una evolución frente a los principios ágiles**. Mientras que estos enfatizan la importancia de la colaboración entre desarrolladores y clientes, **DevOps se centra en la colaboración entre desarrolladores y administradores de sistemas**. Este acercamiento entre ambos ámbitos también permite obtener información de los usuarios a partir del uso que estos hacen del producto en funcionamiento, de manera más fácil y asequible.

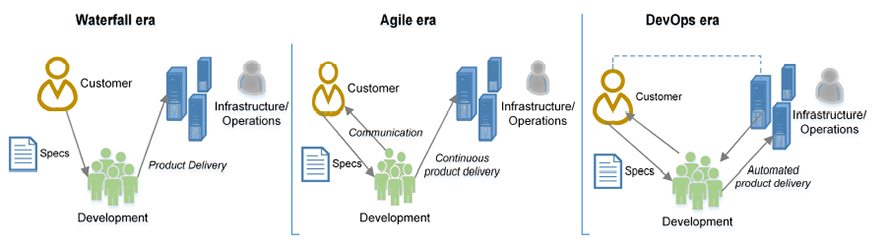


Figura 3. Evolución del modelo «*Waterfall*» al modelo «DevOps». Fuente: Lwakatare (2017)

Origen histórico

Podemos considerar que el movimiento DevOps surgió entre la comunidad de desarrolladores de manera organizada **a partir del año 2007**. En este punto los equipos comenzaron a comprender de manera clara que el modelo tradicional —donde quienes escriben el código (los desarrolladores) se mantienen completamente al margen de aquellos que lo despliegan y mantienen (operaciones de TI)— había quedado obsoleto. Ambos colectivos tenían objetivos diferentes y a menudo opuestos (Atlassian, 2018), direcciones diferentes e incluso ubicaciones físicas distintas.

Podemos establecer la siguiente **cronología breve** (Mezak, 2018):

* **Antes de 2007**: los tradicionales modelos de desarrollo en cascada habían dado paso a nuevas metodologías ágiles en muchos ámbitos de desarrollo. Con ello se perseguía una mejor y más rápida adaptación a los cambios del entorno (agilidad). En paralelo, las operaciones y la gestión de infraestructuras de sistemas de información se estaban transformando, con el surgimiento del *Cloud Computing* y la madurez de los nuevos servicios IaaS (*Infrastructure as a Service*) y PaaS (*Platform as a Service*).
* **Entre 2007 y 2008**: durante la «*Agile Conference*» de 2008 (Agile Alliance, 2018) Andrew Shafer planteó un encuentro de desarrolladores para debatir sobre el tema «Infraestructura Ágil». A él sólo acudió el ingeniero belga Patrick Debois, que llevaba algún tiempo descontento con la manera en que se desarrollaban los proyectos en los que trabajaba. Del diálogo entre ellos y otros desarrolladores surgió el concepto «**administración ágil de sistemas**», que no tuvo demasiado éxito.
* **Año 2009**: durante la **conferencia *Velocity***(O’Reilly, 2018) dos ingenieros de Flickr, John Allspaw y Paul Hammond, realizaron la famosa presentación «10+ Deploys per Day: Dev and Ops Cooperation at Flickr» (Allspaw, 2009), poniendo el dedo en la llaga sobre la necesidad de integración de ambos ámbitos. Patrick Debois se sintió inspirado por esta presentación y animado a organizar la primera conferencia «DevOps Days» en Gante, que tuvo lugar en 2009 (DevOps Days, 2018a). El término DevOps quedó oficialmente acuñado desde entonces.
* **Año 2010**: la conferencia «DevOps Days» se organiza por primera vez en MountainView, California, y el evento se populariza de manera creciente, hasta el punto de que anualmente se organizan decenas de conferencias en todo el mundo (DevOps Days, 2018b).
* **Año 2013**: aparece la novela *The Phoenix Project* (Kim, Behr, & Spafford, 2013), que lleva a la ficción los problemas y soluciones sobre los que había girado la aparición de la cultura DevOps.
* **El futuro de DevOps**: hay quien opina que DevOps es una moda, o más bien un concepto de moda. En cualquier caso, es una manera de enfocar el proceso de desarrollo que busca la mejora continua, y que se ha vuelto imprescindible en el entorno tecnológico actual. DevOps seguirá evolucionando a medida que las metodologías y las herramientas que le sirven de apoyo continúen mejorando y adaptándose a los nuevos proyectos y desarrollos con el paso del tiempo.

Qué no es DevOps

Como hemos visto, DevOps está enfocado a tender un puente entre los desarrolladores y los responsables de operaciones, y viene acompañado de un conjunto de mejores prácticas para la producción de aplicaciones en la nube, siempre orientadas a la integración continua y la liberación frecuente del código. Pero también es importante saber distinguir lo que no es. En concreto (Lent, 2014):

1. **No es un puesto de trabajo, o un rol específico.** Esto puede resultar llamativo, pues a menudo las organizaciones ofertan puestos de trabajo precisamente con esta denominación, «responsable de DevOps», o «experto en DevOps». Aunque es evidente que la implementación de DevOps requiere dominar un conjunto de prácticas e incluso herramientas específicas, como hemos visto se trata más bien de **una cultura que abarca al conjunto de miembros del equipo, y no a un perfil específico diferenciado del resto**.
2. **No es una categoría de herramientas.** DevOps no se ocupa tanto de herramientas específicas, como de la cultura organizacional que permite un desarrollo y despliegue continuo del producto software, favoreciendo la colaboración de todos los implicados en el proceso. Patrick Debois, quien acuñó el término «DevOps» y organizó la primera conferencia «DevOps Days», resalta que **las herramientas juegan un papel fundamental facilitador del proceso, pero que el concepto va mucho más allá.**
3. **No es sinónimo de integración continua.** DevOps surgió en el contexto de trabajo de los equipos ágiles, donde la liberación frecuente de código es una práctica integrada en las diferentes metodologías. Pero **el hecho de que un proyecto realice integraciones continuas no significa necesariamente que exista una cultura DevOps interiorizada**.
4. **No es simplemente un término de moda.** Hay quien opina que DevOps es una moda pasajera, o un término que se repite insistentemente en el ámbito del desarrollo. Pero lo cierto es que, independientemente de la denominación, el cambio de paradigma de desarrollo orientado al despliegue de aplicaciones en la nube refleja una necesidad que no va a desaparecer en el contexto actual, donde cada vez más es necesario adaptarse a los cambios del entorno llegando de manera temprana y manteniendo un adecuado nivel de calidad.

11.3. Prácticas principales

Como hemos visto, DevOps es fundamentalmente un enfoque general del proceso de software que intenta unificar las actividades de desarrollo con las de operación y mantenimiento del sistema, pasando por la involucración del equipo de pruebas, que garantiza la calidad del software entregado. Para que todo esto funcione correctamente es necesario aplicar al menos **siete prácticas principales (figura 4)** (Ott et al., 2017):

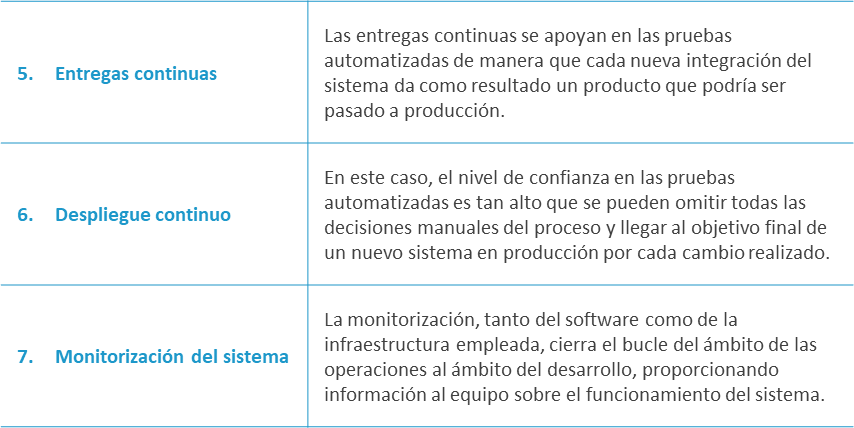
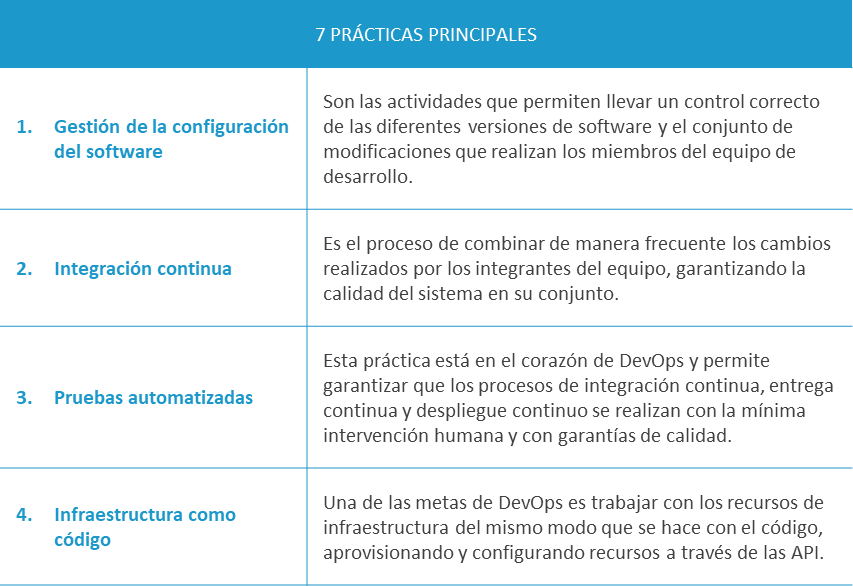


Figura 4. Prácticas principales

Estudiaremos cada una de estas prácticas de manera detenida a continuación.

1. Gestión de configuración del software

La **gestión o administración de la configuración** —*Configuration Management* (CM)—es un término muy amplio que a menudo **se utiliza como sinónimo del control de versiones del código fuente.** Pero también se refiere a la **gestión y el despliegue de diferentes configuraciones a producción**, tanto de software como de hardware.

**En el contexto de DevOps la gestión de configuración hace énfasis en la importancia de mantener un repositorio de código fuente organizado, que permita la automatización y la estandarización de los despliegues a partir del mismo.**

Babich (1986) la define como «el arte de identificar, organizar y controlar las modificaciones que se hacen al software que construirá un equipo de desarrollo», siendo la meta maximizar la productividad del equipo y minimizar los errores. Según Pressman (2010) en este proceso participan **todos los involucrados en el desarrollo**, aunque en ocasiones se crean posiciones especializadas de apoyo.

Es importante tener en cuenta que **el comportamiento de una aplicación no siempre es el mismo en los diferentes entornos** que atraviesa durante el proceso de desarrollo.

Muchos problemas que aparecen en el entorno de producción no pueden detectarse o ser replicados en el entorno de desarrollo, y viceversa.

Para controlar las diferentes versiones de código encontramos una serie de **flujos de trabajo** relacionados con la práctica específica de la gestión de configuración, que se apoyan en los sistemas de control de versiones existentes, como Git.

Los siguientes son los modelos más habituales, presentados de menor a mayor nivel de complejidad (Ott et al., 2017):

* ***Centralized workflow* (flujo centralizado)**. Existe un repositorio central en el que existe una rama de desarrollo denominada «master». Cualquier cambio se realiza directamente sobre ella. Todos los desarrolladores mantienen una copia local del repositorio y, cuando mezclan los cambios realizados, deben resolver a menudo posibles conflictos de manera manual.

Este modelo implica que **las integraciones de código fuente deben realizarse frecuentemente**, para que todo el equipo esté permanentemente actualizado con los últimos cambios. La figura 5 muestra un repositorio con una única rama, que va acumulando todos los cambios realizados.

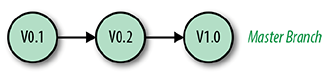


Figura 5. Gestión de versiones del código fuente con flujo de trabajo centralizado.

Fuente: Ott et al. (2017, p. 14)

* ***Feature workflow* (flujo por características)**. En este modelo se crean ramas independientes y dedicadas para el desarrollo de las nuevas características que se van incluyendo en el producto. Esto permite que los desarrolladores trabajen de manera independiente sin interferir con el desarrollo del resto, y sin la necesidad de distribuir los cambios de manera rutinaria y frecuente. Este modelo sirve como base más sólida para otras prácticas, como el despliegue continuo.

La idea se representa en la figura 6 donde, como vemos, además de la rama «master» se crean ramas específicas por cada una de las nuevas características. Todas las ramas de tipo «*feature*» se generan a partir de «master», y cuando la funcionalidad está terminada, se vuelcan los cambios realizados nuevamente sobre master.

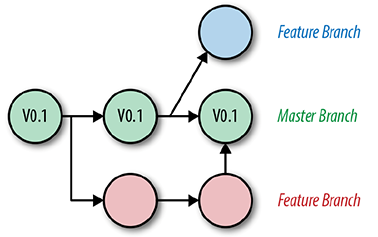


Figura 6. Gestión de versiones del código fuente con flujo de trabajo basado en características (*features*).  
Fuente: Ott et al. (2017, p. 15)

* ***Gitflow workflow* (flujo Gitflow)**. Es un modelo más avanzado que los anteriores, empleado por equipos experimentados. Combina el modelo centralizado y el modelo por características, junto con otra serie de ramas específicas. De ellas, podemos considerar las ramas «master» y «*develop*» como principales, mientras que las ramas de tipo «*feature*», «*release*» y «*hotfix*» sirven de soporte al proceso de desarrollo, y se crean en función de las necesidades (Driessen, 2010; López de la Cruz, 2012). Estas últimas tienen un tiempo de vida limitado, y se eliminan cuando dejen de cumplir su función.
  + ***Master branch* (rama master)**: esta rama principal contiene la historia de versiones oficiales del software. El código se mezcla en esta rama solo cuando se va a liberar una nueva *release*, y cada una de ellas aparece marcada en el repositorio con una etiqueta. Además, en ella solo se mezcla código procedente de una rama «*release*» (que se crean específicamente cuando se va a comenzar el lanzamiento de la nueva *release*), o de una rama «*hotfix*» (que se crea de manera temporal para hacer revisiones menores de una *release*).
  + ***Develop branch* (rama de desarrollo)**: es la rama en la que se produce de manera general el proceso de desarrollo, y donde se van integrando las diferentes características desarrolladas en las ramas temporales de tipo «*feature*». Desde esta rama se realiza el despliegue de integración continua en los servidores de desarrollo.
  + ***Feature branch* (rama de características)**: cada desarrollador crea una de estas ramas para implementar una nueva característica. A diferencia del modelo de flujo por características, en este caso las ramas de características se crean siempre a partir de la rama «*develop*», y cuando se termina la codificación se mezcla el código nuevamente con la rama de desarrollo. Como vemos, «*develop*» es la rama principal de trabajo y de ahí su nombre.
  + ***Release branch* (rama de lanzamiento)**: estas ramas se crean cuando se decide lanzar una nueva *release*, y actúa como paso intermedio entre la rama «*develop*» y la rama «master». Una vez creada una rama «*release*» a partir de «*develop*», solo se admite aplicar sobre ella correcciones de errores, pero no se agregan nuevas características. Cuando se decide que el código en la rama «*release*» está terminado y no contiene errores, se mezcla con «master» y se define una etiqueta que marca la nueva versión de la aplicación que será liberada.
  + ***Hotfix branch* (rama de revisiones o ajustes)**: se trata de ramas temporales, y son las únicas que se crean directamente a partir de «master». Se utilizan para aplicar correcciones o parches que deben ser pasados rápidamente a producción, arreglando errores o vulnerabilidades encontradas en el código de la versión actual. Dan lugar a revisiones de la versión que se encuentra en producción. Cuando se arregla el problema sobre la rama «*hotfix*», el código se mezcla tanto con «master» como con «*develop*», para que los desarrolladores continúen trabajando sobre la versión más actualizada del código que se encuentra en producción.

Todas las ramas del **modelo Gitflow** y sus interacciones aparecen representadas en la figura 7.

Cuando se inicializa el repositorio, se crean tanto la rama «master» como la rama «*develop*». El trabajo de los desarrolladores comienza creando nuevas ramas «*feature*» para implementar nuevas características, que se integran progresivamente sobre «*develop*» a medida que se finalizan. En algún momento se toma la decisión de lanzar una nueva versión del código a producción, creándose una nueva rama «*release*» a partir de «*develop*». Sobre ella únicamente se realiza la corrección de errores y problemas detectados, hasta que finalmente el código se devuelve a «master», creándose una nueva versión que se pasa a producción.

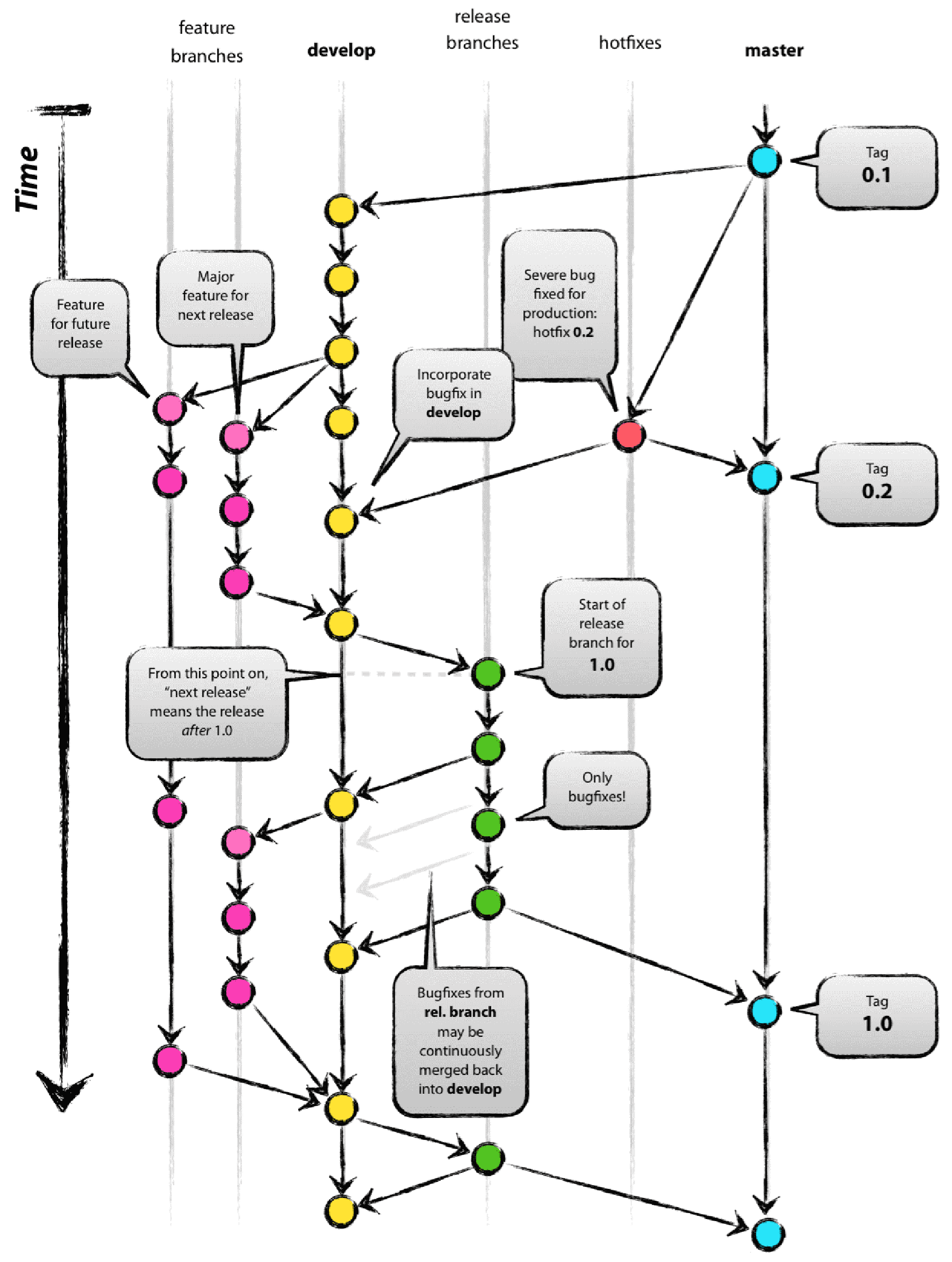
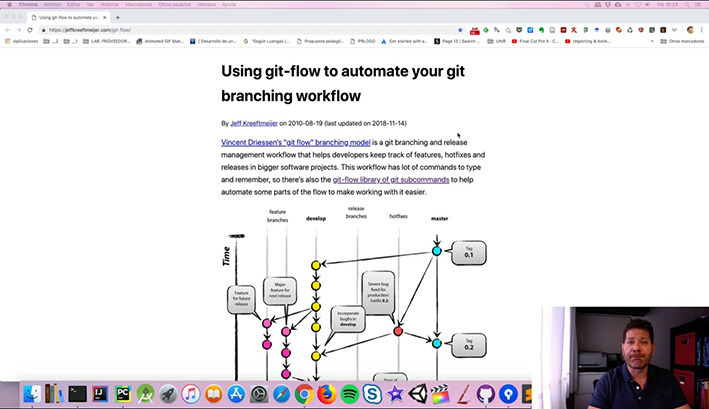


Figura 7. Gestión de versiones del código fuente con el modelo Gitflow.  
Fuente: Driessen (2010)

**Vídeo: Manejo de Gitflow**

En este videotutorial vamos a introducir Gitflow.



Accede al vídeo a través del aula virtual

2. Integración continua

**La integración continua —*Continuous Integration* (CI)— es el proceso que permite la integración del nuevo código generado de manera individual por los desarrolladores en un repositorio común.**

Lo ideal es realizar esta integración con la mayor frecuencia posible y esta actividad sirve como base para otras, como la entrega continua y el despliegue continuo, que veremos más adelante.

La integración continua es un prerrequisito para poder conseguir el despliegue continuo, pero debe trabajar en combinación con otras prácticas, como las pruebas automatizadas, para garantizar que el proceso en su conjunto se realiza con las adecuadas garantías de calidad.

En el contexto de los sistemas de control de versiones, que nos han servido como herramienta de apoyo para explicar la gestión de configuración, la integración continua se representa gráficamente en la figura 8.

A pesar de que un sistema de control de versiones como Git es descentralizado por naturaleza — *Distributed Version Control System* (DVCS)—, todos los desarrolladores pueden definir un repositorio como contenedor canónico del software, sobre el cual todos ellos vuelcan los cambios realizados. Empleando la terminología Git, los desarrolladores hacen «*pull»* para traerse los cambios de un repositorio remoto, y hacen «*push»* para publicar sus modificaciones a las ramas del repositorio remoto. Este repositorio remoto central suele denominarse «*origin*», y es sobre el que se realiza la integración de los cambios realizados por los diferentes desarrolladores.

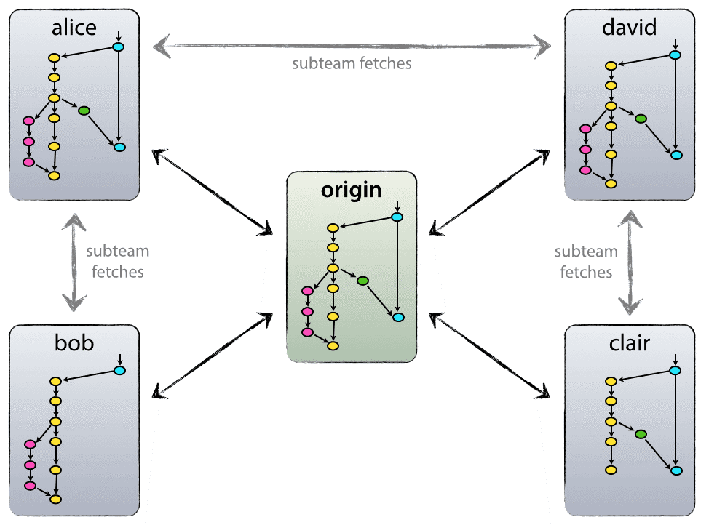


Figura 8. Empleo de un repositorio común de integración continua.  
Fuente: Driessen (2010)

En la figura 8 vemos que también hay definidos dos subequipos de trabajo; el de Alice y Bob, el de Alice y David, y el de Clair y David. Todos ellos tienen definido el repositorio remoto «*origin*» pero, además, Alice podría tener definido un remoto «*bob*», y a su vez Bob realizar «*pull*» y «*push*» sobre el remoto «alice». Cuando uno de los subequipos finaliza las características sobre las que está trabajando, pueden publicar los cambios realizados sobre «*origin*».

3. Pruebas automatizadas

La automatización de tareas está en el corazón de DevOps. **La automatización de las pruebas —*Automated Testing* (AT)— busca adelantarlas todo lo posible dentro de cada ciclo de desarrollo, de manera que posibles errores se detecten de manera temprana.** Esta automatización aumenta la confianza del equipo en que, en cualquier momento, y si así lo desean, podrían pasar una nueva capacidad implementada a producción.

Para comprenderlas nos apoyaremos en la figura 9, que muestra una secuencia habitual y recomendada para la construcción y despliegue de aplicaciones en la nube.

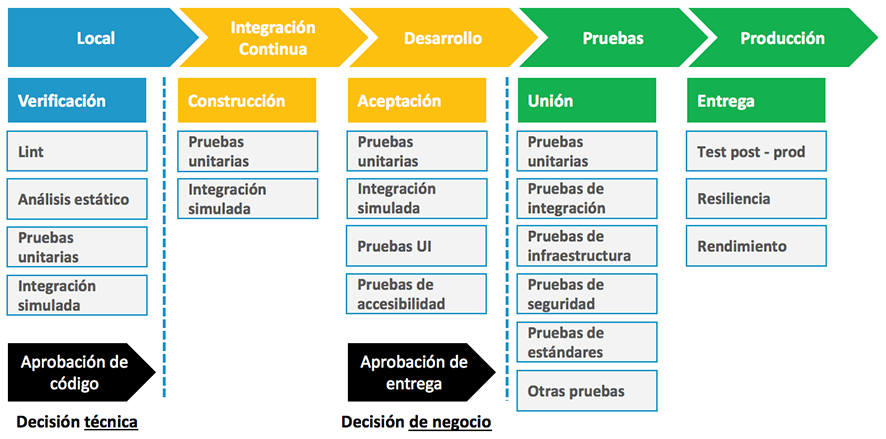


Figura 9. Entornos de desarrollo y conjuntos de pruebas recomendadas.  
Fuente: Elaboración propia a partir de Ott et al. (2017, p. 21), donde por simplicidad se ha eliminado el entorno de producción, que puede ser consultado en la fuente original

Así, DevOps supone **realizar diferentes conjuntos de pruebas en función del entorno en el que estemos trabajando**:

* **Entorno local**. Es el entorno de desarrollo de cada programador, ya se trate de su máquina local o una máquina virtual remota. Es responsabilidad de cada desarrollador corregir los errores que aquí se produzcan. Aquí encontramos las siguientes pruebas básicas:
  + **Lint y análisis estático**. Se trata de analizar el código sin ejecutarlo para detectar posibles errores, problemas de estilo o malas prácticas.
  + **Pruebas unitarias**. Son construidas por cada desarrollador basándose en los criterios de aceptación, y tiene un alcance reducido al código que genera cada programador. Se ejecutan de manera rutinaria y local después de cada cambio introducido y por ello es fundamental que su ejecución será rápida.
  + **Pruebas de integración simulada**. Se simula la interacción con otros componentes y las API, empleando resultados de prueba y componentes específicos que permiten esta simulación («*mock*»).
* **Entorno de integración continua**. Una vez que cada incremento de código es verificado de manera local, y tiene la aprobación del programador, se produce la integración en el repositorio común. Este repositorio se almacena en una máquina diferente a los equipos de desarrollo, y allí se repiten las pruebas unitarias y las pruebas de integración simulada. De esta manera se garantiza que el código en su conjunto, tras las modificaciones, siga funcionando correctamente.
* **Entorno de desarrollo.** En este caso se trata de asegurar que el código está preparado para pasar al entorno de pruebas. En el caso de que falle alguna prueba, el programador correspondiente deberá resolver el error. En este punto se repiten las pruebas unitarias y las pruebas de integración simulada, y se agregan las siguientes:
  + **Pruebas de interfaz de usuario (UI).** Se ejecutan de manera automatizada, desarrollando pruebas funcionales que simulan la interacción del usuario con la aplicación, empleando herramientas como Selenium (2018). Su ejecución es más lenta que la de las pruebas unitarias, y por este motivo no se emplean habitualmente en el entorno de desarrollo.
  + **Pruebas de accesibilidad.** La accesibilidad de las aplicaciones es un aspecto cada vez más importante. En muchos casos es necesaria la compatibilidad con los estándares de la Sección 508 (*United States Access Board*, 2000), o con las guías de accesibilidad para contenido Web del W3C (W3C, 1999, 2008). Aunque la mayoría de estas pruebas son manuales, es posible automatizar en cierta medida el proceso con herramientas como CodeSniffer (CodeSniffer, 2018) o SortSite (PowerMapper, 2018), aunque hay muchas otras (W3C, 2016).
* **Entorno de pruebas**. Una vez que se toma la decisión de negocio de pasar el producto a producción, comienzan las pruebas que permiten garantizar que el sistema en su conjunto funcionará correctamente en el entorno de producción. Aquí se integran todos los componentes, y se repiten las pruebas unitarias. Las pruebas de integración permiten validar que el comportamiento global es el correcto. Además, se incluyen las siguientes pruebas:
  + **Código de infraestructura**. En DevOps, el aprovisionamiento y configuración de los recursos de infraestructura también se realiza a través de código, como veremos, y este código lleva asociadas un conjunto de pruebas.
  + **Pruebas de seguridad**, intentando detectar posibles brechas y vulnerabilidades conocidas.
  + **Pruebas de conformidad con estándares**. Aquí son de utilidad herramientas como OpenSCAP (OpenSCAP, 2018).
  + **Pruebas de resiliencia**, que permiten garantizar la disponibilidad del sistema en condiciones anómalas. Por ejemplo, la herramienta «**Chaos Monkey»** de Netflix (Netflix, 2016/2018) permite eliminar de manera aleatoria instancias de máquinas virtuales, o contenedores, para comprobar la tolerancia a fallos de la arquitectura del sistema.
  + **Pruebas de rendimiento**. Permiten valorar el desempeño global de la aplicación y su escalabilidad.
* **Entorno de producción**. En esta etapa hay una serie de pruebas recomendadas, que están muy relacionadas con la práctica de monitorización continua que veremos más adelante.
  + **Pruebas de posproducción**. Se ejecutan pruebas sobre el sistema y la infraestructura para asegurar que el conjunto tiene un nivel de salud correcto.
  + **Pruebas de resiliencia y rendimiento**.
  + **Pruebas sintéticas**. Se encargan de simular el comportamiento de los usuarios imitando ubicaciones, dispositivos y patrones de uso. Algunas herramientas como «**New Relic»** (2018) permiten definir este tipo de simulaciones.

4. Infraestructura como código

La **infraestructura como código** —*Infrastructure as Code* (IaC)— supone trabajar con los recursos físicos de infraestructura del mismo modo que se hace con el código fuente de las aplicaciones que se ejecutan sobre ellos. Va más allá de la simple automatización de operaciones de configuración de los diferentes entornos empleando *scripts*; **se trata de aprovisionar y configurar recursos de infraestructura a través de las API**. De esta manera, la configuración de los entornos de desarrollo, pruebas o producción se realiza como cualquier otro proceso de desarrollo, y se emplean las mismas prácticas (control de versiones, pruebas unitarias, etc.).

La implementación de esta práctica puede adoptar dos enfoques (Ott et al., 2017):

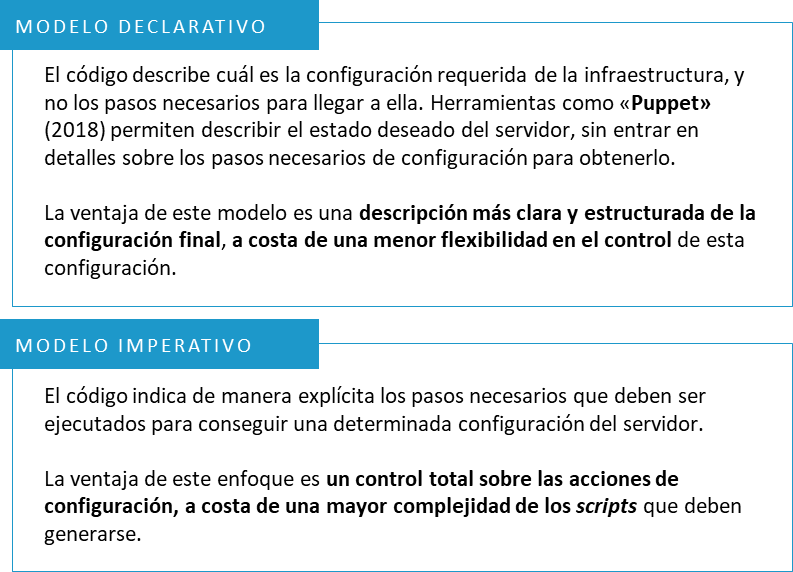


Figura 10. Enfoques de la infraestructura como código

5. Entregas continuas

**La entrega continua significa que cualquier cambio realizado da como resultado un sistema susceptible de ser pasado a producción tan pronto como las pruebas automatizadas lo validen.** Ello implica la toma de dos decisiones, como vemos en la figura 9:

1. Una **decisión técnica**, que aprueba el paso al entorno de integración.
2. Una **decisión de negocio**, que acepta los cambios realizados y permite proseguir a los entornos de producción.

DevOps es una reacción cultural a la lenta entrega de nuevas versiones y características, incluso cuando se empleaba un enfoque ágil. Los principios ágiles permiten la construcción incremental de software, pero las liberaciones están limitadas al final de cada iteración, donde se ha terminado el trabajo planificado y el *Product Owner,* o responsable de negocio, da el visto bueno para el despliegue. **DevOps** elimina esta última barrera que limita los tiempos de entrega de nuevas versiones y **da el poder al *Product Owner* para decidir en qué momento pasar a producción una nueva historia de usuario**, aunque el resto de las historias de la iteración no estén completas.

Para que las entregas continuas sean posibles,

el equipo ha debido alcanzar una gran madurez de la aplicación

de las cuatro prácticas anteriores (CM, CI, AT, e IaC).

6. Despliegue continuo

Mientras que las entregas continuas hacen referencia a la potencialidad de pasar a producción cualquier incremento realizado, pero están sujeta a decisiones procedentes de negocio, **el despliegue continuo —*Continuous Deployment* (CD)—implica una mayor confianza en la calidad de las pruebas realizadas; tan alta, que hace que cualquier cambio se pase a producción tan pronto como el código supere las pruebas asociadas.**

En términos prácticos, esto significa que la decisión técnica de la figura 9 se puede eliminar y la decisión de negocio, que da paso a los entornos de producción, se vuelve opcional.

El **despliegue continuo es el objetivo final de las organizaciones más eficientes**, pero ello no quiere decir que todas las operaciones se realicen sin ningún tipo de intervención manual. Ello dependerá del grado de desarrollo de las pruebas automatizadas y de posibles valoraciones y pruebas de aceptación por parte del cliente o el *Product Owner*.

7. Monitorización continua

La monitorización continua —*Continuous Monitoring* (CM)— cierra el bucle de realimentación conectando operaciones y desarrollo.

El equipo de desarrollo es partícipe del proceso de monitorización de la aplicación en producción, labor que tradicionalmente estaba reservada a personal de operaciones, administradores de sistemas y analistas. De esta manera, se reduce el tiempo entre la posible detección de un problema y su corrección.

Las labores de monitorización no se piensan y diseñan *a posteriori,* sino que la definición de las herramientas necesarias se define desde el comienzo del proyecto e implica al equipo de desarrollo.

Podemos identificar **varios tipos de monitorización** (Ott et al., 2017):

* De infraestructura
* De rendimiento de aplicaciones
* De logs
* De seguridad

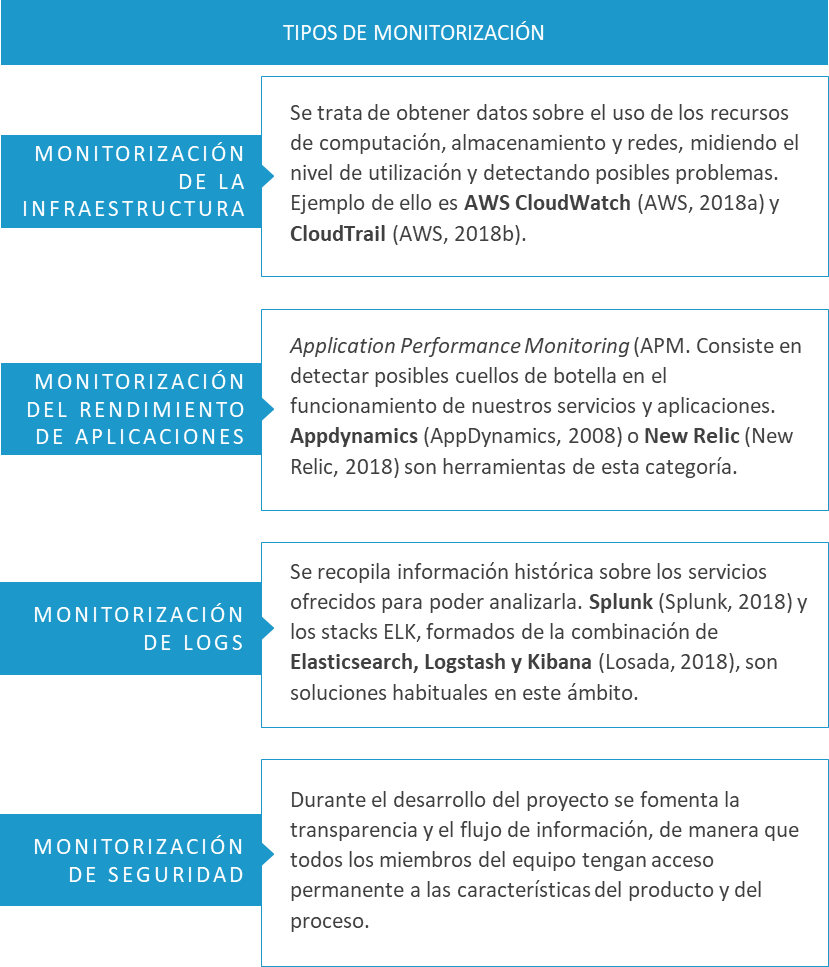


Figura 11. Tipos de monitorización

11.4. Referencias bibliográficas

Agile Alliance. (2018). Agile Conferences Calendar [Web]. Recuperado de <https://www.agilealliance.org/events/calendars/agile-conferences-calendar/>

Allspaw, J. (2009). 10+ Deploys Per Day: Dev and Ops Cooperation at Flickr. Technology. Recuperado de <https://www.slideshare.net/jallspaw/10-deploys-per-day-dev-and-ops-cooperation-at-flickr>

AppDynamics. (2008). Application Performance Monitoring & Management [Web]. Recuperado de <https://www.appdynamics.com/>

Atlassian. (2018). ¿Qué es DevOps? [Artículo web]. Recuperado de <https://es.atlassian.com/devops>

AWS. (2018a). Amazon CloudWatch: Monitoreo de infraestructuras y aplicaciones [Web]. Recuperado de <https://aws.amazon.com/es/cloudwatch/>

AWS. (2018b). AWS CloudTrail: Realice un seguimiento de la actividad de los usuarios y el uso de las API [Web]. Recuperado de <https://aws.amazon.com/es/cloudtrail/>

Babich, W. A. (1986). *Software Configuration Management: Coordination for Team Productivity*. Reading, Mass: Addison-Wesley.

Bass, L., Weber, I. & Zhu, L. (2015). *DevOps: A Software Architect’s Perspective* (1ª ed.). New York: Addison-Wesley Professional.

Bello, C. (2017, enero 30). ¿Sabes realmente qué es DevOps? [Blog]. Recuperado de <https://www.xeridia.com/blog/sabes-realmente-que-es-devops>

CodeSniffer. (2018). HTML\_CodeSniffer [Web]. Recuperado de <https://squizlabs.github.io/HTML_CodeSniffer/>

DevOps Days. (2018a). About Devopsdays [Web]. Recuperado de <https://www.devopsdays.org/about/>

DevOps Days. (2018b). DevOpsDays Conferences [Web]. Recuperado de <https://www.devopsdays.org/>

Driessen, V. (2010). A successful Git branching model [Blog personal]. Recuperado de <http://nvie.com/posts/a-successful-git-branching-model/>

Kim, G., Behr, K. & Spafford, G. (2013). *The Phoenix Project: A Novel about It, Devops, and Helping Your Business Win*. Portland, OR: IT Revolution Press.

Lent, J. (2014). Definición de DevOps: mejor explicamos lo que no es [Artículo Web]. Recuperado de <https://searchdatacenter.techtarget.com/es/consejo/Definicion-de-DevOps-mejor-explicamos-lo-que-no-es>

López de la Cruz, J. (2012, noviembre 24). Una buena manera de afrontar la ramificación (branching) en Git [Post]. Recuperado de <https://jesuslc.com/2012/11/24/una-buena-manera-de-afrontar-la-ramificacion-branching-en-git/>

Losada, S. (2018, julio 30). ¿Qué es ELK? ElasticSearch, Logstash y Kibana [Artículo Web]. Recuperado de <https://openwebinars.net/blog/que-es-elk-elasticsearch-logstash-y-kibana/>

Lwakatare, L. E. (2017). *DevOps adoption and implementation in software development practice: concept, practices, benefits and challenges* (Tesis doctoral). University of Oulu,

Finland.

Mezak, S. (2018, enero 25). The Origins of DevOps: What’s in a Name? [Blog]. Recuperado de <https://devops.com/the-origins-of-devops-whats-in-a-name/>

Netflix. (2018). Chaos Monkey is a resiliency tool that helps applications tolerate random instance failures. Recuperado de <https://github.com/Netflix/chaosmonkey>

Netflix. (2018). Security Monkey monitors AWS, GCP, OpenStack, and GitHub orgs for assets and their changes over time. Recuperado de <https://github.com/Netflix/security_monkey>

New Relic. (2018). New Relic. Real-time insights for modern software [Web]. Recuperado de <https://newrelic.com/>

OpenSCAP. (2018). Audit, Fix and be Merry [Web]. Recuperado de <https://www.open-scap.org/>

O’Reilly. (2018). Systems Engineering & DevOps Training: O’Reilly Velocity. Recuperado de <https://conferences.oreilly.com/velocity>

Ott, B., Pham, J. & Saker, H. (2017). *Enterprise DevOps Playbook*. O’Reilly.

PowerMapper. (2018). SortSite-Accessibility Checker and Validator. Recuperado de <https://www.powermapper.com/products/sortsite/checks/accessibility-checks/>

Pressman, R. S. (2010). *Ingeniería del software: un enfoque práctico* (7ª ed.). México: McGraw-Hill Educación.

Puppet. (2018). Infrastructure as code [Web]. Recuperado de <https://puppet.com/solutions/infrastructure-as-code>

Salmanson, J. (2015). DevOps in Practice [Presentación]. Recuperado de <https://www.census.gov/fedcasic/fc2015/ppt/22_salmanson.pdf>

Selenium. (2018). Selenium–Web Browser Automation [Web]. Recuperado de <https://www.seleniumhq.org/>

Splunk. (2018). SIEM, AIOps, Application Management, Log Management, Machine Learning, and Compliance [Web]. Recuperado de <https://www.splunk.com/>

United States Access Board. (2000). Section 508 Standards for Electronic and Information Technology [Informe]. Recuperado de <https://www.access-board.gov/guidelines-and-standards/communications-and-it/about-the-section-508-standards/section-508-standards>

W3C. (1999). Web Content Accessibility Guidelines 1.0 [Guía]. Recuperado de <https://www.w3.org/TR/WCAG10/>

W3C. (2008). Web Content Accessibility Guidelines (WCAG) 2.0 [Guía]. Recuperado de <https://www.w3.org/TR/WCAG20/>

W3C. (2016). Web Accessibility Evaluation Tools List [Guía]. Recuperado de <https://www.w3.org/WAI/ER/tools/>

A fondo

Lección magistral: Principales prácticas en DevOps

En esta lección magistral comenzaremos explicando el concepto de DevOps, que supone un cambio de paradigma en el desarrollo de aplicaciones respecto del modelo tradicional. Además, explicaremos las siete prácticas principales que conviene tener en cuenta cuando se adopta este modelo, y señalaremos las diferencias entre la integración, la entrega y el despliegue continuos.



Accede a la lección magistral a través del aula virtual.

**The (Short) History of DevOps**

En este vídeo Damon Edwars nos recuerda los orígenes históricos de DevOps como movimiento social que intenta acercar a las comunidades de desarrollo y de operaciones.

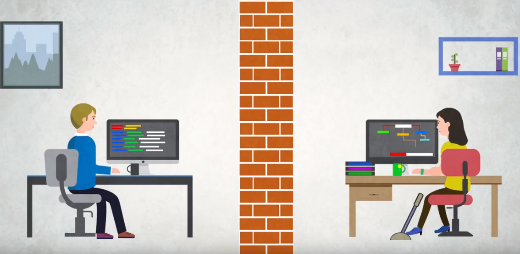


Accede al vídeo a través del aula virtual o desde la siguiente dirección web:

https://youtu.be/o7-IuYS0iSE

**DevOps en menos de 3 minutos**

En este breve vídeo encontrarás una introducción al concepto de DevOps, como acercamiento entre los mundos de desarrollo y operaciones. Fomenta la comunicación, permite compartir activos, automatiza procesos y facilita la retroalimentación continua.

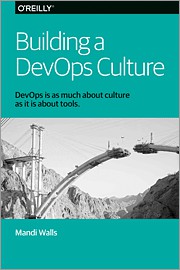


Accede al vídeo a través del aula virtual o desde la siguiente dirección web:

<https://youtu.be/p-bOnV8FRMQ>

Building a DevOps Culture

Walls, M. (s. f.). *Building a DevOps Culture*. O’Reilly.



La editorial O’Reilly mantiene una página específicamente dedicada a compartir pequeños libros gratuitos relacionados con DevOps y otras temáticas afines. En concreto, este libro de Mandi Walls presenta una introducción al concepto de DevOps, enfatizando en que se trata de mucho más que la utilización de un conjunto de herramientas, que supone cambiar la cultura de la organización de manera que los equipos de trabajo integren participantes tanto del mundo del desarrollo como del mundo de las operaciones.

Accede al documento a través del aula virtual o desde la siguiente dirección web:

https://www.oreilly.com/webops-perf/free/building-devops-culture.csp

DevOpsDays

DevOpsDays es un conjunto de conferencias técnicas que cubren aspectos desde el desarrollo del software hasta las operaciones de infraestructura IT, y la intersección entre ambos dominios. La primera conferencia fue organizada por Patrick Debois en Gante, Bélgica, en 2009. Su página web contiene información actualizada sobre futuras conferencias, y es un buen punto de partida para tomar contacto con la comunidad de profesionales interesados en estos temas.



Accede a la web a través del aula virtual o desde la siguiente dirección web:

https://www.devopsdays.org/

Introducción al modelo de control de versiones de código fuente Gitflow

En este artículo encontrarás una traducción bastante libre del artículo original de Vincent Driessen, que explica los fundamentos del control de versiones de código fuente propuesto por del modelo Gitflow.

Accede al documento a través del aula virtual o desde la siguiente dirección web:

<https://jesuslc.com/2012/11/24/una-buena-manera-de-afrontar-la-ramificacion-branching-en-git/>

Test

1. ¿Cuál de las siguientes personas acuñó el término DevOps y organizó la primera conferencia DevOps Days?

A. Bill Gates

B. Tim O’Reilly

C. Patrick Debois

D. Andrew Shafer

1. ¿Cuál de las siguientes opciones describe mejor lo que es DevOps?

A. Un perfil de especialización concreto para un miembro del equipo de trabajo.

B. Una cultura de la organización que acerca los ámbitos de desarrollo y operaciones.

C. Un conjunto de herramientas que permiten a los desarrolladores colaborar de manera más cercana con los administradores de sistemas.

D. El nombre de una conferencia internacional.

1. ¿Cuáles de los siguientes hechos están en el origen de DevOps y justifican su aparición?

A. La popularización de las metodologías ágiles de desarrollo.

B. La popularización de las aplicaciones en la nube que requieren de actualizaciones continuas.

C. La madurez de los servicios IaaS que permiten manipular la infraestructura de manera parecida al desarrollo de código, a través de las API.

D. Todas las respuestas anteriores son correctas.

1. ¿Qué es «The Phoenix Project»?

A. El primer proyecto en el que se aplicó DevOps de manera clara.

B. Una novela de ficción sobre el surgimiento de DevOps como movimiento cultural en el ámbito del software.

C. La quinta novela de la saga de Harry Potter.

D. Ninguna de las respuestas anteriores es correcta.

1. ¿Qué prácticas DevOps son necesarias para alcanzar la entrega continua?

A. La gestión de la configuración, la monitorización del sistema y la infraestructura como código.

B. El despliegue continuo, la monitorización del sistema y las pruebas automatizadas.

C. La gestión de configuración, el despliegue continuo y la monitorización del sistema.

D. La gestión de configuración, la integración continua, las pruebas automatizadas y la infraestructura como código.

1. ¿Cuántas ramas hay en un repositorio de software que sigue un modelo de flujo de trabajo por características?

A. Una rama «master» y una rama de tipo «*feature*» por cada característica implementada.

B. Solo una rama «master».

C. Tantas ramas como versiones del código se pasan a producción.

D. Solamente dos. Una rama «master» y una rama de tipo «*feature*».

1. ¿Para qué sirve una rama «*hotfix*» en el modelo de repositorio Gitflow?

A. Para desarrollar las nuevas características del producto.

B. Para aplicar arreglos o correcciones de errores sobre una rama «*development*»

C. Para aplicar arreglos o correcciones de errores sobre la rama «master».

D. Ninguna de las respuestas anteriores es correcta.

1. ¿Cómo suele denominarse el repositorio de código donde se integran las modificaciones de código realizadas por los diferentes desarrolladores en una práctica de integración continua?

A. *Origin*.

B. Master.

C. *Development*.

D. Todas las respuestas anteriores son correctas.

1. ¿Qué aportación tiene la práctica de entregas continuas respecto del enfoque general de las metodologías ágiles?

A. Permite que desde un punto de vista de negocio se pueda tomar la decisión de pasar el producto a producción en cualquier momento de la iteración.

B. Da mayores herramientas al *Product Owner* para monitorizar la calidad del código de los desarrolladores.

C. Convierte la decisión de negocio de pasar la aplicación a producción en un paso opcional, al estar toda la cadena completamente automatizada.

D. Todas las respuestas anteriores son correctas.

1. ¿Qué es un stack ELK?

A. Una solución que permite la recolección, centralización y explotación de datos procedentes de logs.

B. La combinación de tres productos: Elasticsearch, Logstash y Kibana.

C. Una solución que permite la monitorización de una aplicación de software en producción.

D. Todas las respuestas anteriores son correctas.