Elegir una estrategia de ramificación de Git para entornos de múltiples cuentas DevOps

AWS Orientación prescriptiva

AWS Orientación prescriptiva: Elegir una estrategia de ramificación de Git para entornos de múltiples cuentas DevOps

Table of Contents

Elegir una estrategia de ramificación de Git para entornos de múltiples cuentas DevOps

Amazon Web Services (colaboradores)

Febrero de 2024 (historial del documento)

Pasar a un enfoque basado en la nube y ofrecer soluciones de software AWS puede ser transformador. Es posible que requiera cambios en el proceso del ciclo de vida del desarrollo de software. Por lo general, Cuentas de AWS se utilizan varios durante el proceso de desarrollo en el Nube de AWS. Elegir una estrategia de ramificación de Git compatible que combine con tus DevOps procesos es esencial para el éxito. Elegir la estrategia de ramificación de Git adecuada para tu organización te ayuda a comunicar de forma concisa DevOps los estándares y las mejores prácticas entre los equipos de desarrollo. La ramificación de Git puede ser sencilla en un solo entorno, pero puede resultar confusa cuando se aplica en varios entornos, como entornos sandbox, de desarrollo, de pruebas, de puesta en escena y de producción. Tener varios entornos aumenta la complejidad de la implementación. DevOps

Esta guía proporciona diagramas visuales de las estrategias de ramificación de Git que muestran cómo una organización puede implementar un proceso de múltiples cuentas DevOps . Las guías visuales ayudan a los equipos a entender cómo combinar sus estrategias de ramificación de Git con sus DevOps prácticas. El uso de un modelo de ramificación estándar, como GitHub Gitflow, Flow o Trunk, para gestionar el repositorio de código fuente ayuda a los equipos de desarrollo a alinear su trabajo. Estos equipos también pueden utilizar los recursos de formación estándar de Git en Internet para comprender e implementar esos modelos y estrategias.

Para conocer las DevOps mejores prácticas al respecto AWS, consulte la DevOpsGuía de AWS Well-Architected. Al revisar esta guía, utilice la diligencia debida para seleccionar la estrategia de ramificación adecuada para su organización. Es posible que algunas estrategias se adapten mejor a su caso de uso que otras.

Objetivos

Esta guía forma parte de una serie de documentos sobre cómo elegir e implementar estrategias de DevOps ramificación para organizaciones con múltiples Cuentas de AWS sucursales. Esta serie está diseñada para ayudarlo a aplicar la estrategia que mejor se adapte a sus requisitos, objetivos y

mejores prácticas desde el principio, a fin de optimizar su experiencia en la Nube de AWS industria. Esta guía no contiene scripts DevOps ejecutables porque varían según el motor de integración y entrega continuas (CI/CD) y los marcos tecnológicos que utilice su organización.

Esta guía explica las diferencias entre tres estrategias comunes de ramificación de Git: GitHub Flow, Gitflow y Trunk. Las recomendaciones de esta guía ayudan a los equipos a identificar una estrategia de ramificación que se ajuste a los objetivos de su organización. Tras revisar esta guía, deberías poder elegir una estrategia de ramificación para tu organización. Tras elegir una estrategia, puedes usar uno de los siguientes patrones para ayudarte a implementar esa estrategia con tus equipos de desarrollo:

- Implemente una estrategia de ramificación troncal para entornos con varias cuentas DevOps
- Implemente una estrategia de ramificación GitHub de Flow para entornos con varias cuentas DevOps
- Implementa una estrategia de ramificación de Gitflow para entornos con varias cuentas DevOps

Es importante tener en cuenta que lo que funciona para una organización, equipo o proyecto puede no ser adecuado para otras. La elección entre las estrategias de ramificación de Git depende de varios factores, como el tamaño del equipo, los requisitos del proyecto y el equilibrio deseado entre la colaboración, la frecuencia de integración y la gestión de las versiones.

Prácticas de uso CI/CD

AWS recomienda implementar la integración y la entrega continuas (CI/CD), which is the process of automating the software release lifecycle. It automates much or all of the manual DevOps processes that are traditionally required to get new code from development into production. A CI/CD pipeline encompasses the sandbox, development, testing, staging, and production environments. In each environment, the CI/CD pipeline provisions any infrastructure that is needed to deploy or test the code. By using CI/CD, development teams can make changes to code that are then automatically tested and deployed. CI/CDlas canalizaciones también proporcionan control y protección a los equipos de desarrollo). Imponen la coherencia, los estándares, las mejores prácticas y los niveles mínimos de aceptación para la aceptación y el despliegue de las funciones. Para obtener más información, consulte Practicar la integración continua y la entrega continua en AWS.

Todas las estrategias de ramificación que se analizan en esta guía son adecuadas para CI/CD practices. The complexity of the CI/CD pipeline increases with the complexity of the branching strategy. For example, Gitflow is the most complex branching strategy discussed in this guide. CI/

CD pipelines for this strategy require more steps (such as for compliance reasons), and they must support multiple, simultaneous production releases. Using CI/CD also becomes more important as the complexity of the branching strategy increases. This is because CI/CD establecer barreras y mecanismos para los equipos de desarrollo que impidan que los desarrolladores eludan intencional o involuntariamente el proceso definido.

AWS ofrece un conjunto de servicios para desarrolladores diseñados para ayudarle a crear canalizaciones de CI/CD. Por ejemplo, AWS CodePipelinees un servicio de entrega continua totalmente gestionado que le ayuda a automatizar sus procesos de lanzamiento para obtener actualizaciones rápidas y fiables de las aplicaciones y la infraestructura. AWS CodeBuildcompila el código fuente, ejecuta pruebas y produce paquetes de ready-to-deploy software. Para obtener más información, consulte Developer Tools on AWS.

Comprensión de los DevOps entornos

Para comprender las estrategias de ramificación, debe comprender el propósito y las actividades que se llevan a cabo en cada entorno. El establecimiento de varios entornos le ayuda a separar las actividades de desarrollo en etapas, a supervisarlas y a evitar el lanzamiento involuntario de funciones no aprobadas. Puede tener uno o más Cuentas de AWS en cada entorno.

La mayoría de las organizaciones tienen varios entornos descritos para su uso. Sin embargo, la cantidad de entornos puede variar según la organización y según las políticas de desarrollo de software. En esta serie de documentación se supone que tiene los siguientes cinco entornos comunes que abarcan su proceso de desarrollo, aunque es posible que tengan nombres diferentes:

- Sandbox: un entorno en el que los desarrolladores escriben código, cometen errores y realizan trabajos de prueba de concepto.
- Desarrollo: un entorno en el que los desarrolladores integran su código para confirmar que todo funciona como una aplicación única y cohesiva.
- Pruebas: un entorno en el que se llevan a cabo equipos de control de calidad o pruebas de aceptación. Los equipos suelen realizar pruebas de rendimiento o integración en este entorno.
- Preparación: un entorno de preproducción en el que se valida que el código y la infraestructura funcionan según lo esperado en circunstancias equivalentes a las de producción. Este entorno está configurado para ser lo más parecido posible al entorno de producción.
- Producción: un entorno que gestiona el tráfico de sus usuarios finales y clientes.

En esta sección se describe cada entorno en detalle. También describe los pasos de creación, los pasos de implementación y los criterios de salida de cada entorno para que pueda continuar con el siguiente. La siguiente imagen muestra estos entornos en secuencia.

Temas de esta sección:

- Entorno sandbox
- Entorno de desarrollo

- Entorno de pruebas
- Entorno de puesta en escena
- Entorno de producción

Entorno sandbox

El entorno sandbox es donde los desarrolladores escriben código, cometen errores y realizan trabajos de prueba de concepto. Puede realizar la implementación en un entorno sandbox desde una estación de trabajo local o mediante un script en una estación de trabajo local.

Acceso

Los desarrolladores deben tener acceso completo al entorno sandbox.

Pasos de construcción

Los desarrolladores ejecutan la compilación manualmente en sus estaciones de trabajo locales cuando están preparados para implementar cambios en el entorno sandbox.

1. Usa git-secrets (GitHub) para buscar información confidencial
2. Borra el código fuente
3. Cree y compile el código fuente, si corresponde
4. Realice pruebas unitarias
5. Realice un análisis de cobertura de código
6. Realizar un análisis de código estático
7. Cree la infraestructura como código (IaC)
8. Realice un análisis de seguridad de IaC
9. Extraiga licencias de código abierto
10. Publica artefactos de construcción

Pasos de implementación

Si utilizas los modelos Gitflow o Trunk, los pasos de implementación se inician automáticamente cuando una `feature` rama se crea correctamente en el entorno sandbox. Si utilizas el modelo

GitHub Flow, debes realizar manualmente los siguientes pasos de despliegue. Los siguientes son los pasos de implementación en el entorno sandbox:

1. Descargue los artefactos publicados
2. Realice el versionado de la base de datos
3. Realice el despliegue de iAC
4. Realice pruebas de integración

Expectativas antes de pasar al entorno de desarrollo

- Construcción exitosa de la `feature` sucursal en un entorno sandbox
- Un desarrollador implementó y probó la función manualmente en el entorno sandbox

Entorno de desarrollo

El entorno de desarrollo es donde los desarrolladores integran su código para garantizar que todo funcione como una aplicación cohesiva. En Gitflow, el entorno de desarrollo contiene las últimas funciones incluidas en la solicitud de fusión y están listas para su lanzamiento. En las estrategias GitHub Flow y Trunk, el entorno de desarrollo se considera un entorno de pruebas y el código base puede ser inestable e inadecuado para su despliegue en producción.

Acceso

Asigne los permisos de acuerdo con el principio de privilegios mínimos. El privilegio mínimo es la práctica recomendada de seguridad que consiste en conceder los permisos mínimos necesarios para realizar una tarea. Los desarrolladores deberían tener menos acceso al entorno de desarrollo que al entorno sandbox.

Pasos de construcción

Al crear una solicitud de fusión para la `develop` rama (Gitflow) o la `main` rama (Trunk o GitHub Flow), se inicia automáticamente la compilación.

1. Usa git-secrets (GitHub) para buscar información confidencial
2. Borra el código fuente

3. Cree y compile el código fuente, si corresponde
4. Realice pruebas unitarias
5. Realice un análisis de cobertura de código
6. Realizar un análisis de código estático
7. Construye iAC
8. Realice un análisis de seguridad de IaC
9. Extraiga licencias de código abierto

Pasos de implementación

Si utilizas el modelo de Gitflow, los pasos de despliegue se inician automáticamente cuando una `develop` rama se ha creado correctamente en el entorno de desarrollo. Si utilizas el modelo GitHub Flow o el modelo Trunk, los pasos de despliegue se inician automáticamente cuando se crea una solicitud de fusión en la `main` sucursal. Los siguientes son los pasos de implementación en el entorno de desarrollo:

1. Descargue los artefactos publicados desde los pasos de construcción
2. Realice el versionado de la base de datos
3. Realice el despliegue de iAC
4. Realice pruebas de integración

Expectativas antes de pasar al entorno de pruebas

- Creación e implementación satisfactorios de la `develop` rama (Gitflow) o la `main` sucursal (Trunk o GitHub Flow) en el entorno de desarrollo
- Las pruebas unitarias se aprueban al 100%
- Construcción exitosa de iAC
- Los artefactos de despliegue se crearon correctamente
- Un desarrollador ha realizado una verificación manual para confirmar que la función funciona según lo esperado

Entorno de pruebas

El personal de control de calidad (QA) utiliza el entorno de pruebas para validar las características. Aprueban los cambios una vez finalizadas las pruebas. Cuando lo aprueban, la sucursal pasa al siguiente entorno, el de puesta en escena. En Gitflow, este entorno y otros anteriores solo están disponibles para su implementación desde `release` sucursales. Una `release` sucursal se basa en una `develop` rama que contiene las funciones planificadas.

Acceso

Asigne los permisos según el principio del privilegio mínimo. Los desarrolladores deberían tener menos acceso al entorno de pruebas que al entorno de desarrollo. El personal de control de calidad necesita permisos suficientes para probar la función.

Pasos de construcción

El proceso de compilación en este entorno solo se aplica a las correcciones de errores cuando se utiliza la estrategia de Gitflow. Al crear una solicitud de fusión para la `bugfix` sucursal, se inicia automáticamente la compilación.

1. Usa git-secrets (GitHub) para buscar información confidencial
2. Borra el código fuente
3. Cree y compile el código fuente, si corresponde
4. Realice pruebas unitarias
5. Realice un análisis de cobertura de código
6. Realizar un análisis de código estático
7. Construye iAC
8. Realice un análisis de seguridad de IaC
9. Extraiga licencias de código abierto

Pasos de implementación

Inicie automáticamente el despliegue de la `release` sucursal (Gitflow) o de la `main` sucursal (Trunk o GitHub Flow) en el entorno de prueba tras el despliegue en el entorno de desarrollo. Los siguientes son los pasos de implementación en el entorno de pruebas:

1. Implemente la `release` rama (Gitflow) o la `main` rama (Trunk o GitHub Flow) en el entorno de prueba
2. Haga una pausa para la aprobación manual por parte del personal designado
3. Descarga los artefactos publicados
4. Realice el versionado de la base de datos
5. Realice el despliegue de iAC
6. Realice pruebas de integración
7. Realice pruebas de rendimiento
8. Aprobación de control de calidad

Expectativas antes de pasar al entorno de ensayo

- Los equipos de desarrollo y control de calidad han realizado suficientes pruebas para satisfacer los requisitos de su organización.
- El equipo de desarrollo ha resuelto los errores descubiertos a través de una `bugfix` sucursal.

Entorno de puesta en escena

El entorno de ensayo está configurado para ser el mismo que el entorno de producción. Por ejemplo, la configuración de los datos debe ser similar en alcance y tamaño a las cargas de trabajo de producción. Utilice el entorno de ensayo para comprobar que el código y la infraestructura funcionan según lo esperado. Este entorno también es la opción preferida para los casos de uso empresarial, como las vistas previas o las demostraciones para clientes.

Acceso

Asigne los permisos de acuerdo con el principio de privilegios mínimos. Los desarrolladores deben tener el mismo acceso al entorno de ensayo que al entorno de producción.

Pasos de construcción

Ninguna. Los mismos artefactos que se usaron en el entorno de prueba se reutilizan en el entorno de ensayo.

Pasos de implementación

Inicie automáticamente el despliegue de la `release` rama (Gitflow) o la `main` sucursal (Trunk o GitHub Flow) en el entorno de ensayo tras la aprobación y el despliegue en el entorno de prueba. Los siguientes son los pasos de implementación en el entorno provisional:

1. Implemente la `release` rama (Gitflow) o la `main` rama (Trunk o GitHub Flow) en el entorno de ensayo
2. Haga una pausa para la aprobación manual por parte del personal designado
3. Descarga los artefactos publicados
4. Realice el versionado de la base de datos
5. Realice el despliegue de iAC
6. (Opcional) Realice pruebas de integración
7. (Opcional) Realice pruebas de carga
8. Obtenga la aprobación de los responsables de desarrollo, control de calidad, productos o empresas necesarios

Expectativas antes de pasar al entorno de producción

- Se implementó correctamente una versión equivalente a la producción en el entorno de ensayo
- (Opcional) Las pruebas de integración y carga se realizaron correctamente

Entorno de producción

El entorno de producción respalda el producto lanzado y gestiona datos reales de clientes reales. Se trata de un entorno protegido al que se le asigna el acceso con el mínimo privilegio y el acceso elevado solo debe permitirse mediante un proceso de excepción auditado durante un período de tiempo limitado.

Acceso

En el entorno de producción, los desarrolladores deben tener acceso limitado y de solo lectura a la consola de administración de AWS. Por ejemplo, los desarrolladores deberían poder acceder a los datos de registro para las operaciones diarias. Todas las versiones para producción deben estar sujetas a un paso de aprobación antes de su implementación.

Pasos de construcción

Ninguna. Los mismos artefactos que se usaron en los entornos de prueba y puesta en escena se reutilizan en el entorno de producción.

Pasos de implementación

Inicie automáticamente el despliegue de la `release` sucursal (Gitflow) o la `main` sucursal (Trunk o GitHub Flow) en el entorno de producción tras la aprobación y el despliegue en el entorno provisional. Los siguientes son los pasos de implementación en el entorno de producción:

1. Implemente la `release` sucursal (Gitflow) o la `main` rama (Trunk o GitHub Flow) en el entorno de producción
2. Haga una pausa para la aprobación manual por parte del personal designado
3. Descarga los artefactos publicados
4. Realice el versionado de la base de datos
5. Realice el despliegue de iAC

Mejores prácticas para el desarrollo basado en Git

Para adoptar con éxito el desarrollo basado en Git, es importante seguir un conjunto de mejores prácticas que promuevan la colaboración, mantengan la calidad del código y respalden la integración y la entrega continuas (CI/CD). Además de las mejores prácticas de esta guía, consulte la Guía de AWS DevOps Well-Architected. Las siguientes son algunas de las mejores prácticas clave para el desarrollo basado en Git en AWS:

- Mantenga los cambios pequeños y frecuentes: anime a los desarrolladores a realizar cambios o funciones pequeños e incrementales. Esto reduce el riesgo de conflictos de fusión y facilita la identificación y la solución rápida de los problemas.
- Utilice los conmutadores de funciones: para gestionar la publicación de funciones incompletas o experimentales, utilice las opciones o los indicadores de funciones. Esto te ayuda a ocultar, activar o desactivar funciones específicas en producción sin que ello afecte a la estabilidad de la rama principal.
- Mantenga un conjunto de pruebas sólido: un conjunto de pruebas completo y bien mantenido es crucial para detectar los problemas de manera temprana y verificar que la base de código se mantenga estable. Invierta en la automatización de las pruebas y priorice la corrección de las pruebas fallidas.
- Adopte la integración continua: utilice herramientas y prácticas de integración continua para crear, probar e integrar automáticamente los cambios de código en la `develop` rama (Gitflow) o en la `main` rama (Trunk o GitHub Flow). Esto le ayuda a detectar los problemas de forma temprana y agiliza el proceso de desarrollo.
- Realice revisiones del código: fomente las revisiones del código por pares para mantener la calidad, compartir conocimientos y detectar posibles problemas antes de que se integren en la `main` rama. Usa solicitudes de extracción u otras herramientas de revisión de código para facilitar este proceso.
- Supervisa y corrige compilaciones defectuosas: cuando una compilación se interrumpe o las pruebas fallan, prioriza la solución del problema lo antes posible. Esto mantiene la `develop` rama (Gitflow) o la `main` rama (Trunk o GitHub Flow) en un estado liberable y minimiza el impacto en otros desarrolladores.
- Comunícate y colabora: promueve la comunicación abierta y la colaboración entre los miembros del equipo. Asegúrese de que los desarrolladores estén al tanto del trabajo en curso y de los cambios que se están realizando en el código base.

- Refactoriza de forma continua: refactoriza periódicamente el código base para mejorar su mantenibilidad y reducir la deuda técnica. Anime a los desarrolladores a dejar el código en un estado mejor del que lo encontraron.
- Usa ramas de corta duración para tareas complejas: para tareas más grandes o complejas, usa ramas de corta duración (también conocidas como ramas de tareas) para trabajar en los cambios. Sin embargo, asegúrate de que la vida útil de las ramas sea corta, normalmente menos de un día. Vuelve a combinar los cambios en la `develop` rama (Gitflow) o en la `main` rama (Trunk o GitHub Flow) lo antes posible. Las fusiones y revisiones más pequeñas y frecuentes son más fáciles de procesar para un equipo que una solicitud de fusión grande.
- Capacite y apoye al equipo: brinde capacitación y apoyo a los desarrolladores que son nuevos en el desarrollo basado en Git o que requieren orientación para adoptar sus mejores prácticas.

Estrategias de ramificación de Git

En esta guía, ordenadas de menor a mayor complejidad, se describen en detalle las siguientes estrategias de Git-based ramificación:

- Troncal: Trunk-based el desarrollo es una práctica de desarrollo de software en la que todos los desarrolladores trabajan en una sola rama, normalmente denominada `main` rama `trunk` o. La idea detrás de este enfoque es mantener el código base en un estado de publicación continua integrando los cambios de código con frecuencia y confiando en las pruebas automatizadas y la integración continua.
- GitHub Flow: GitHub Flow es un flujo de trabajo ligero y basado en ramas que fue desarrollado por. GitHub Se basa en la idea de ramas de corta duración`feature`. Cuando una función está completa y lista para su implementación, la función se fusiona en la `main` rama.
- Gitflow: con un enfoque de Gitflow, el desarrollo se completa en ramas de funciones individuales. Tras la aprobación, se fusionan `feature` las ramas en una rama de integración que suele tener un nombre. `develop` Cuando se han acumulado suficientes funciones en la `develop` rama, se crea una `release` rama para implementar las funciones en los entornos superiores.

Cada estrategia de ramificación tiene ventajas y desventajas. Si bien todas utilizan los mismos entornos, no todas utilizan las mismas sucursales ni los mismos pasos de aprobación manual. En esta sección de la guía, revise cada estrategia de ramificación en detalle para familiarizarse con sus matices y poder evaluar si se ajusta al caso de uso de su organización.

Temas de esta sección:

- Estrategia de ramificación troncal
- GitHub Estrategia de ramificación de flujos
- Estrategia de ramificación de Gitflow

Estrategia de ramificación troncal

Trunk-based el desarrollo es una práctica de desarrollo de software en la que todos los desarrolladores trabajan en una sola rama, normalmente denominada `main` rama `trunk` o. La idea detrás de este enfoque es mantener el código base en un estado de publicación continua integrando

los cambios de código con frecuencia y confiando en las pruebas automatizadas y la integración continua.

En el desarrollo basado en enlaces troncales, los desarrolladores envían sus cambios a la `main` rama varias veces al día, con el objetivo de realizar actualizaciones pequeñas e incrementales. Esto permite ciclos de retroalimentación rápidos, reduce el riesgo de conflictos de fusión y fomenta la colaboración entre los miembros del equipo. La práctica enfatiza la importancia de un conjunto de pruebas bien mantenido porque se basa en pruebas automatizadas para detectar posibles problemas de manera temprana y garantizar que la base de código permanezca estable y liberable.

Trunk-based El desarrollo suele contrastarse con el desarrollo basado en funciones (también conocido como ramificación de funciones o desarrollo impulsado por funciones), en el que cada nueva función o corrección de errores se desarrolla en su propia rama dedicada, separada de la rama principal. La elección entre el desarrollo basado en funciones o el desarrollo basado en funciones depende de factores como el tamaño del equipo, los requisitos del proyecto y el equilibrio deseado entre la colaboración, la frecuencia de integración y la gestión de las versiones.

Para obtener más información sobre la estrategia de ramificación de Trunk, consulta los siguientes recursos:

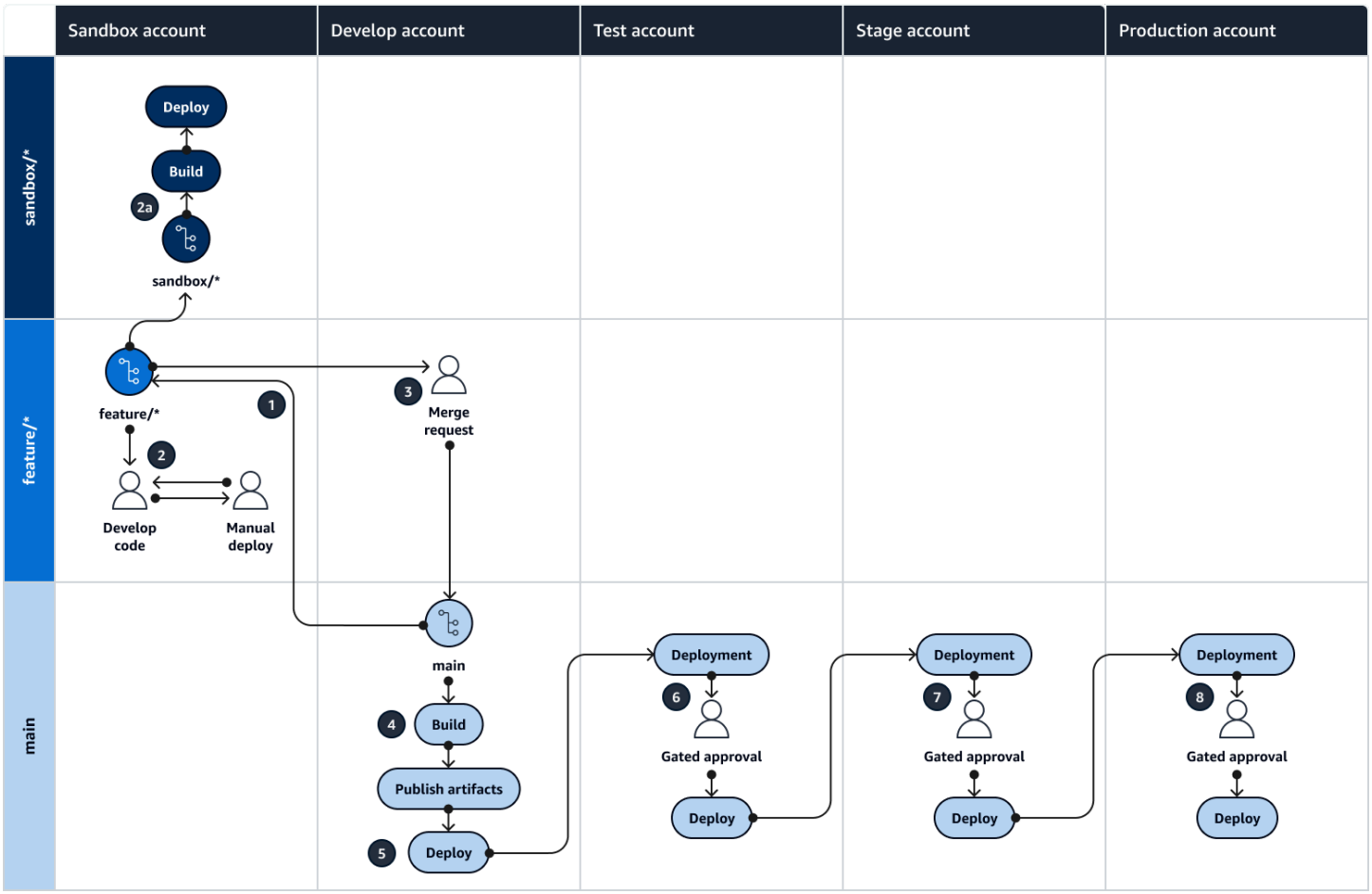
- Implemente una estrategia de ramificación troncal para DevOps entornos con varias cuentas (AWS orientación prescriptiva)
- Introducción al Trunk-Based desarrollo (sitio web de desarrollo basado en troncales)

Temas de esta sección:

Resumen visual de la estrategia de Trunk

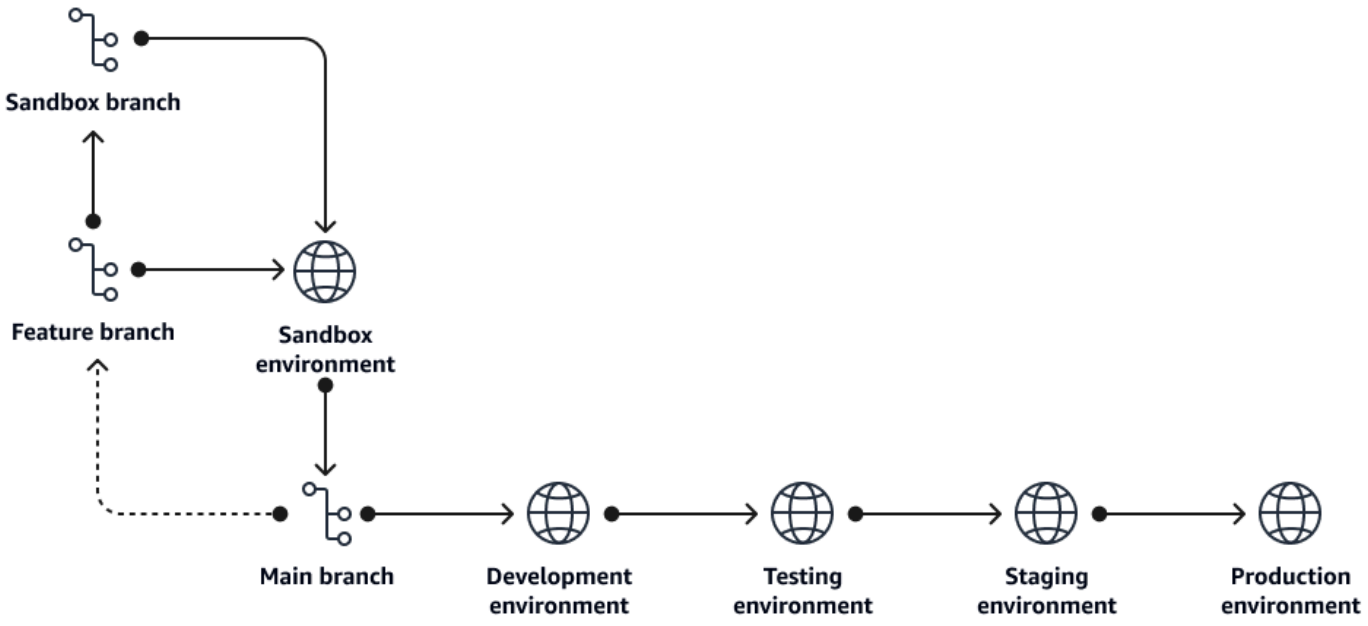
El siguiente diagrama se puede utilizar como un cuadro de Punnett (Wikipedia) para entender la estrategia de ramificación de Trunk. Alinee las ramas del eje vertical con los AWS entornos del eje horizontal para determinar qué acciones realizar en cada escenario. Los números encerrados en un círculo te guían por la secuencia de acciones representada en el diagrama. Este diagrama muestra el flujo de trabajo de desarrollo de una estrategia de ramificación troncal, desde una `feature`

sucursal en un entorno sandbox hasta la versión de producción de la rama. `main` Para obtener más información sobre las actividades que se producen en cada entorno, consulte DevOps los entornos en esta guía.



La estrategia de ramificaciones en un tronco

Una estrategia de ramificación troncal suele tener las siguientes ramas.



rama de característica

Desarrollas funciones o creas una revisión en una `feature` rama. Para crear una `feature` rama, se ramifica desde la `main` rama. Los desarrolladores iteran, confirman y prueban el código de una `feature` rama. Cuando una función está completa, el desarrollador la promociona. Solo hay dos caminos hacia adelante desde una `feature` rama:

- Incorpórate a la `sandbox` rama
- Crea una solicitud de fusión en la `main` sucursal

Convención de nomenclatura:	`feature/<story number>_<developer initials>_<descriptor>`
Ejemplo de convención de nomenclatura:	`feature/123456_MS_Implement_Feature_A`

rama sandbox

Esta rama es una rama troncal no estándar, pero es útil para el desarrollo de CI/CD tuberías. La `sandbox` rama se utiliza principalmente para los siguientes fines:

- Realice un despliegue completo en el entorno sandbox mediante las canalizaciones CI/CD
- Desarrolle y pruebe una canalización antes de enviar solicitudes de fusión para realizar pruebas completas en un entorno inferior, como el de desarrollo o las pruebas.

`Sandbox`las sucursales son de naturaleza temporal y están destinadas a ser de corta duración. Deben suprimirse una vez finalizadas las pruebas específicas.

Convención de nomenclatura:	`sandbox/<story number>_<developer initials>_<descriptor>`
Ejemplo de convención de nomenclatura:	`sandbox/123456_MS_Test_Pipeline_Deploy`

rama principal

La `main` rama siempre representa el código que se está ejecutando en producción. El código se ramifica, se desarrolla y`main`, a continuación, se fusiona nuevamente con`main`. Las implementaciones desde `main` pueden dirigirse a cualquier entorno. Para evitar que se eliminen, habilite la protección de sucursales para la `main` sucursal.

Convención de nomenclatura:	`main`

rama de hotfix

No hay una `hotfix` rama dedicada en un flujo de trabajo basado en enlaces troncales. Los hotfixes utilizan ramas. `feature`

Ventajas y desventajas de la estrategia Trunk

La estrategia de ramificación de Trunk es ideal para equipos de desarrollo más pequeños y maduros que tienen sólidas habilidades de comunicación. También funciona bien si tiene versiones continuas y continuas de funciones para la aplicación. No es la solución adecuada si sus equipos de desarrollo son grandes o están fragmentados, o si tiene programadas versiones amplias de funciones. En este modelo se producirán conflictos de fusión, así que tenga en cuenta que la resolución de conflictos

de fusión es una habilidad clave. Todos los miembros del equipo deben estar capacitados en consecuencia.

Ventajas

Trunk-based el desarrollo ofrece varias ventajas que pueden mejorar el proceso de desarrollo, agilizar la colaboración y mejorar la calidad general del software. Las siguientes son algunas de las principales ventajas:

- Bucles de retroalimentación más rápidos: con el desarrollo basado en enlaces troncales, los desarrolladores integran sus cambios de código con frecuencia, a menudo varias veces al día. Esto permite obtener información más rápida sobre posibles problemas y ayuda a los desarrolladores a identificar y solucionar los problemas con mayor rapidez de lo que lo harían en un modelo de desarrollo basado en funciones.
- Reducción de los conflictos de fusión: en el desarrollo basado en enlaces troncales, el riesgo de conflictos de fusión grandes y complicados se minimiza porque los cambios se integran de forma continua. Esto ayuda a mantener una base de código más limpia y reduce el tiempo dedicado a resolver conflictos. La resolución de conflictos puede llevar mucho tiempo y ser propensa a errores en el desarrollo basado en funciones.
- Colaboración mejorada: Trunk-based el desarrollo alienta a los desarrolladores a trabajar juntos en la misma sucursal, lo que promueve una mejor comunicación y colaboración dentro del equipo. Esto puede conducir a una resolución de problemas más rápida y a una dinámica de equipo más cohesiva.
- Revisiones de código más sencillas: dado que los cambios de código son más pequeños y más frecuentes en el desarrollo basado en troncos, puede resultar más fácil realizar revisiones exhaustivas del código. Los cambios más pequeños suelen ser más fáciles de entender y revisar, lo que permite identificar de forma más eficaz los posibles problemas y mejoras.
- Integración y entrega continuas: el Trunk-based desarrollo respalda los principios de integración y entrega continuas (CI/CD). Al mantener el código base en un estado que se pueda publicar e integrar los cambios con frecuencia, los equipos pueden adoptar CI/CD prácticas con mayor facilidad, lo que se traduce en ciclos de implementación más rápidos y en una mejor calidad del software.
- Calidad del código mejorada: con la integración, las pruebas y las revisiones del código frecuentes, el desarrollo basado en troncos puede contribuir a mejorar la calidad general del código. Los desarrolladores pueden detectar y solucionar los problemas con mayor rapidez, lo que reduce la probabilidad de que la deuda técnica se acumule con el tiempo.

- Estrategia de ramificación simplificada: Trunk-based el desarrollo simplifica la estrategia de ramificación al reducir el número de sucursales de larga duración. Esto puede facilitar la administración y el mantenimiento del código base, especialmente en el caso de proyectos o equipos de gran tamaño.

Desventajas

Trunk-based el desarrollo tiene algunas desventajas, que pueden afectar al proceso de desarrollo y a la dinámica del equipo. Los siguientes son algunos inconvenientes notables:

- Aislamiento limitado: dado que todos los desarrolladores trabajan en la misma rama, todos los miembros del equipo pueden ver sus cambios de forma inmediata. Esto puede provocar interferencias o conflictos, provocar efectos secundarios no deseados o interrumpir la compilación. Por el contrario, el desarrollo basado en funciones aísla mejor los cambios para que los desarrolladores puedan trabajar de forma más independiente.
- Mayor presión sobre las pruebas: Trunk-based el desarrollo se basa en la integración continua y las pruebas automatizadas para detectar los problemas rápidamente. Sin embargo, este enfoque puede ejercer una gran presión sobre la infraestructura de pruebas y requiere un conjunto de pruebas bien mantenido. Si las pruebas no son exhaustivas o fiables, pueden producirse problemas no detectados en la rama principal.
- Menor control sobre las versiones: Trunk-based el objetivo del desarrollo es mantener el código base en un estado de publicación continua. Si bien esto puede ser ventajoso, puede que no siempre sea adecuado para proyectos con calendarios de lanzamiento estrictos o aquellos que requieren que funciones específicas se publiquen juntas. Feature-basedel desarrollo proporciona un mayor control sobre cuándo y cómo se lanzan las funciones.
- Pérdida de código: dado que los desarrolladores integran constantemente los cambios en la rama principal, el desarrollo basado en enlaces troncales puede provocar una mayor pérdida de código. Esto puede dificultar que los desarrolladores realicen un seguimiento del estado actual del código base y puede generar confusión al intentar comprender el efecto de los cambios recientes.
- Requiere una cultura de equipo sólida: Trunk-based el desarrollo exige un alto nivel de disciplina, comunicación y colaboración entre los miembros del equipo. Esto puede ser difícil de mantener, especialmente en equipos más grandes o cuando se trabaja con desarrolladores que tienen menos experiencia con este enfoque.
- Retos de escalabilidad: a medida que crece el tamaño del equipo de desarrollo, la cantidad de cambios de código que se integran en la rama principal puede aumentar rápidamente. Esto puede

provocar interrupciones de compilación y errores en las pruebas más frecuentes, lo que dificulta mantener el código base en un estado que pueda publicarse.

GitHub Estrategia de ramificación de flujos

GitHub Flow es un flujo de trabajo ligero y basado en sucursales que fue desarrollado por. GitHub GitHub El flujo se basa en la idea de ramas de funciones de corta duración que se fusionan en la rama principal cuando la función está completa y lista para su implementación. Los principios clave de GitHub Flow son:

- La ramificación es ligera: los desarrolladores pueden crear ramas de funciones para su trabajo con solo unos pocos clics, lo que mejora la capacidad de colaborar y experimentar sin afectar a la rama principal.
- Despliegue continuo: los cambios se implementan tan pronto como se fusionan en la rama principal, lo que permite una rápida retroalimentación e iteración.
- Solicitudes de fusión: los desarrolladores utilizan las solicitudes de fusión para iniciar un proceso de discusión y revisión antes de fusionar sus cambios en la rama principal.

Para obtener más información sobre GitHub Flow, consulta los siguientes recursos:

- Implemente una estrategia GitHub de ramificación de Flow para DevOps entornos con varias cuentas (guíaAWS prescriptiva)
- GitHub Flow Quickstart (documentación) GitHub
- ¿Por qué GitHub Flow? (Sitio web GitHub de Flow)

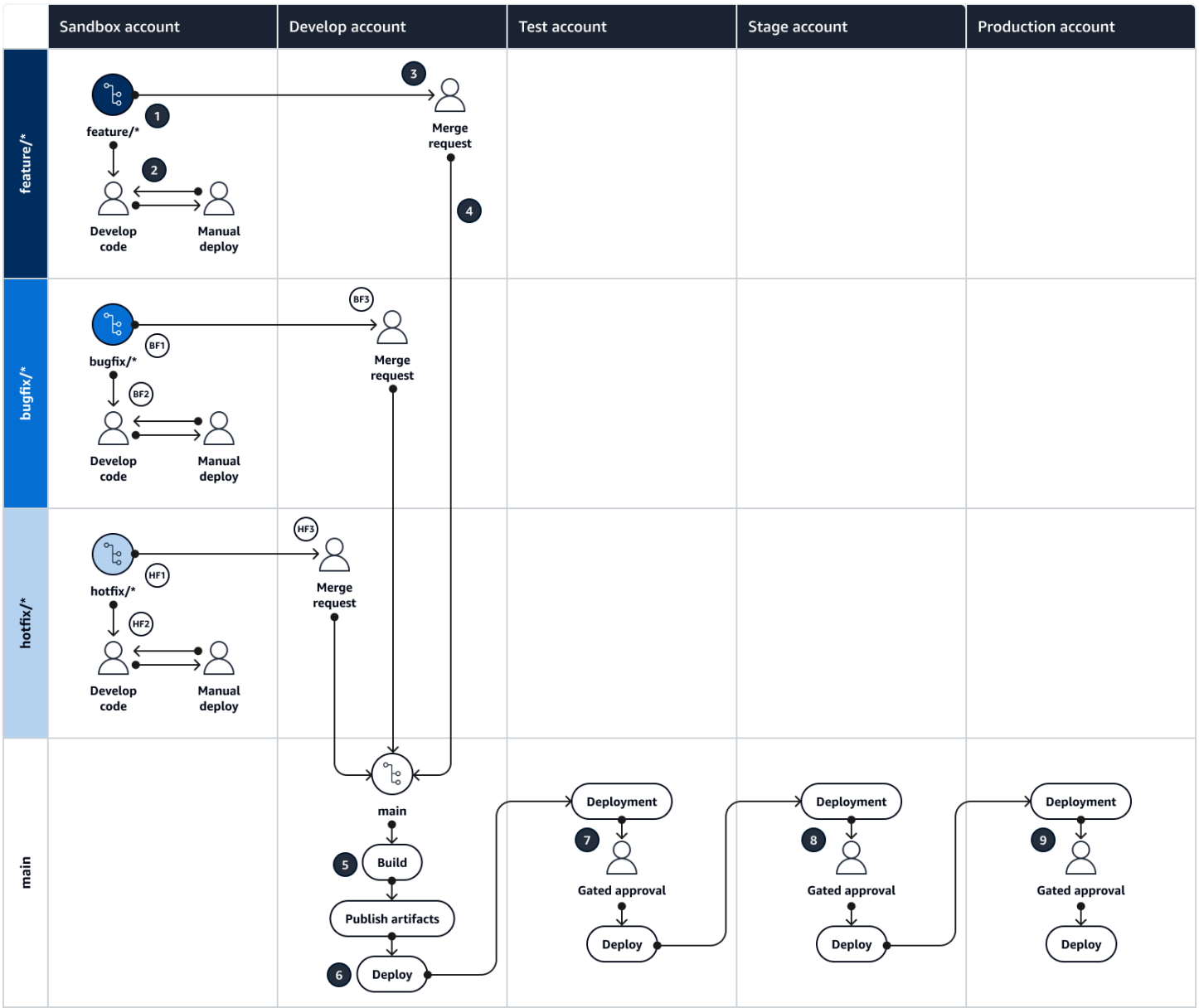
Temas de esta sección:

- Descripción visual de la estrategia GitHub Flow
- Las ramas en una estrategia de GitHub flujo
- Ventajas y desventajas de la estrategia GitHub Flow

Descripción visual de la estrategia GitHub Flow

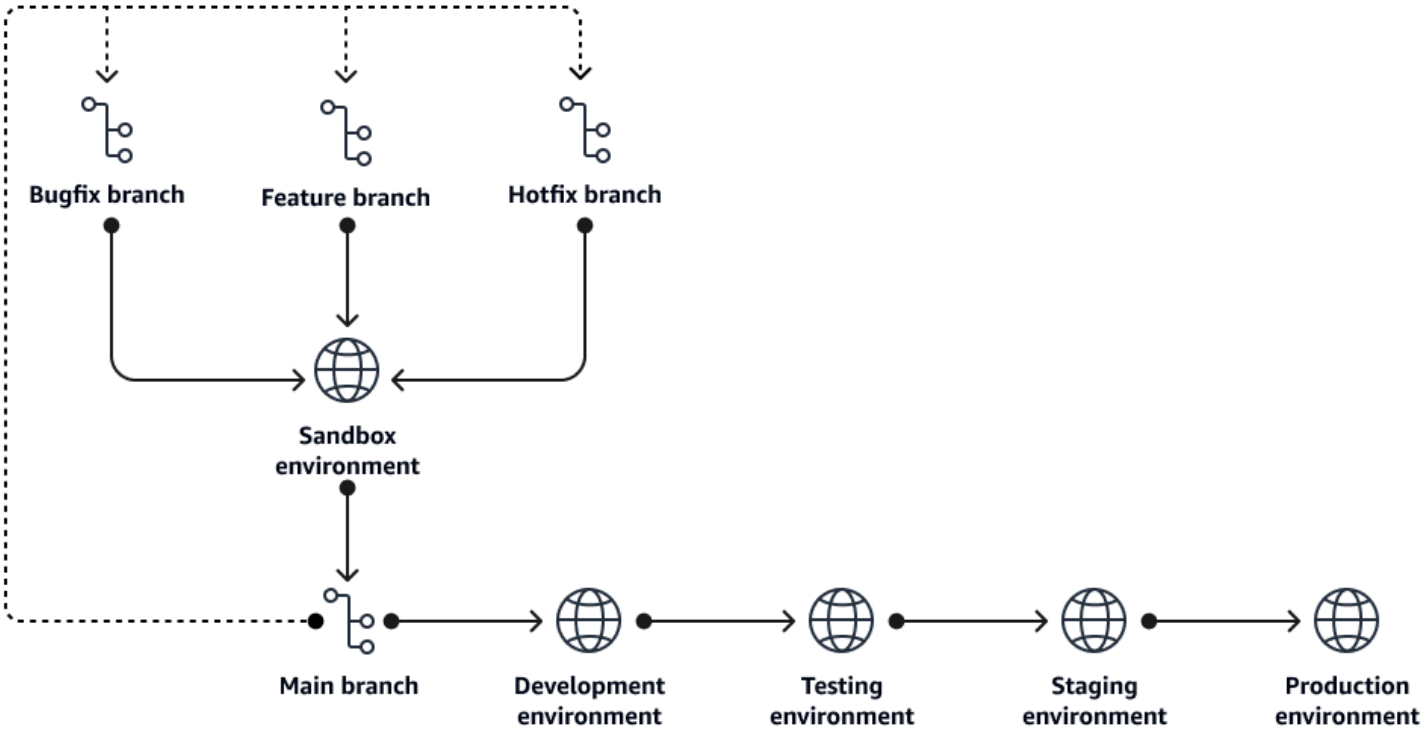
El siguiente diagrama se puede utilizar como un cuadro de Punnett para entender la estrategia de ramificación de GitHub Flow. Alinee las ramas del eje vertical con los AWS entornos del eje

horizontal para determinar qué acciones realizar en cada escenario. Los números encerrados en un círculo te guían por la secuencia de acciones representada en el diagrama. Este diagrama muestra el flujo de trabajo de desarrollo de una estrategia de ramificación de GitHub Flow, desde una rama de funciones en un entorno sandbox hasta la versión de producción de la rama principal. Para obtener más información sobre las actividades que se producen en cada entorno, consulte DevOps los entornos en esta guía.



Las ramas en una estrategia de GitHub flujo

Una estrategia GitHub de ramificación de Flow suele tener las siguientes ramas.



rama de característica

Desarrollas funciones en las `feature` sucursales. Para crear una `feature` rama, se ramifica a partir de la `main` rama. Los desarrolladores iteran, confirman y prueban el código de la `feature` rama. Cuando una función está completa, el desarrollador la promociona creando una solicitud de fusión para`main`.

Convención de nomenclatura:	`feature/<story number>_<developer initials>_<descriptor>`
Ejemplo de convención de nomenclatura:	`feature/123456_MS_Implement_Feature_A`

rama de corrección de errores

La `bugfix` rama se usa para solucionar problemas. Estas ramas se ramifican a partir de la `main` rama. Una vez que la corrección del error se haya probado en entornos aislados o en cualquiera de los entornos inferiores, se puede promover a entornos superiores fusionándola mediante una

solicitud de fusión. `main` Esta es una convención de nomenclatura sugerida para la organización y el seguimiento. Este proceso también se puede gestionar mediante una rama de funciones.

Convención de nomenclatura:	`bugfix/<ticket number>_<developer initials>_<descriptor>`
Ejemplo de convención de nomenclatura:	`bugfix/123456_MS_Fix_Problem_A`

rama de hotfix

La `hotfix` sucursal se utiliza para resolver problemas críticos de alto impacto con una demora mínima entre el personal de desarrollo y el código implementado en producción. Estas sucursales se ramifican fuera de la `main` sucursal. Una vez que la revisión se haya probado en entornos aislados o en alguno de los entornos inferiores, se puede ascender a entornos superiores si se fusiona mediante una solicitud de fusión. `main` Esta es una convención de nomenclatura sugerida para la organización y el seguimiento. Este proceso también se puede gestionar mediante una rama de funciones.

Convención de nomenclatura:	`hotfix/<ticket number>_<developer initials>_<descriptor>`
Ejemplo de convención de nomenclatura:	`hotfix/123456_MS_Fix_Problem_A`

rama principal

La `main` rama siempre representa el código que se está ejecutando en producción. El código se fusiona en la `main` rama desde `feature` las ramas mediante solicitudes de combinación. Para evitar que se eliminen y para evitar que los desarrolladores envíen el código directamente a ella`main`, habilita la protección de la `main` rama en cuestión.

Convención de nomenclatura:	`main`

Ventajas y desventajas de la estrategia GitHub Flow

La estrategia de ramificación de Github Flow es ideal para equipos de desarrollo más pequeños y maduros que tienen sólidas habilidades de comunicación. Esta estrategia es adecuada para los equipos que desean implementar la entrega continua y está bien respaldada por CI/CD los motores comunes. GitHub Flow es ligero, no tiene demasiadas reglas y es capaz de apoyar a los equipos que se mueven rápidamente. No es adecuado si tus equipos tienen que seguir procesos estrictos de cumplimiento o publicación. Los conflictos de fusión son comunes en este modelo y es probable que ocurran con frecuencia. La resolución de conflictos de fusión es una habilidad clave, y debes capacitar a todos los miembros del equipo en consecuencia.

Ventajas

GitHub Flow ofrece varias ventajas que pueden mejorar el proceso de desarrollo, agilizar la colaboración y mejorar la calidad general del software. Los siguientes son algunos de los beneficios clave:

- Flexible y ligero: GitHub Flow es un flujo de trabajo ligero y flexible que ayuda a los desarrolladores a colaborar en proyectos de desarrollo de software. Permite una iteración y experimentación rápidas con una complejidad mínima.
- Colaboración simplificada: GitHub Flow proporciona un proceso claro y simplificado para gestionar el desarrollo de funciones. Fomenta cambios pequeños y específicos que se pueden revisar y combinar rápidamente, lo que mejora la eficiencia.
- Control de versiones claro: con GitHub Flow, cada cambio se realiza en una rama independiente. Esto establece un historial de control de versiones claro y rastreable. Esto ayuda a los desarrolladores a rastrear y comprender los cambios, revertirlos si es necesario y mantener una base de código confiable.
- Integración continua perfecta: GitHub Flow se integra con herramientas de integración continua. La creación de solicitudes de cambios puede iniciar procesos automatizados de pruebas e implementación. Las herramientas de CI te ayudan a probar minuciosamente los cambios antes de integrarlos en la `main` rama, lo que reduce el riesgo de introducir errores en el código base.
- Retroalimentación rápida y mejora continua: GitHub Flow fomenta un ciclo de retroalimentación rápido al promover revisiones frecuentes del código y debates a través de solicitudes de selección de información. Esto facilita la detección temprana de problemas, promueve el intercambio de conocimientos entre los miembros del equipo y, en última instancia, conduce a una mayor calidad del código y a una mejor colaboración dentro del equipo de desarrollo.

- Reversiones y reversiones simplificadas: en el caso de que un cambio de código introduzca un error o un problema inesperado, GitHub Flow simplifica el proceso de revertir o revertir el cambio. Al tener un historial claro de confirmaciones y ramificaciones, es más fácil identificar y revertir los cambios problemáticos, lo que ayuda a mantener una base de código estable y funcional.
- Curva de aprendizaje ligera: GitHub Flow puede ser más fácil de aprender y adoptar que Gitflow, especialmente para los equipos que ya están familiarizados con Git y los conceptos de control de versiones. Su sencillez y su intuitivo modelo de ramificación hacen que sea accesible para desarrolladores con distintos niveles de experiencia, lo que reduce la curva de aprendizaje asociada a la adopción de nuevos flujos de trabajo de desarrollo.
- Desarrollo continuo: GitHub Flow permite a los equipos adoptar un enfoque de implementación continua al permitir la implementación inmediata de cada cambio tan pronto como se incorpore a la `main` sucursal. Este proceso simplificado elimina las demoras innecesarias y garantiza que las últimas actualizaciones y mejoras estén disponibles rápidamente para los usuarios. Esto se traduce en un ciclo de desarrollo más ágil y con mayor capacidad de respuesta.

Desventajas

Si bien GitHub Flow ofrece varias ventajas, también es importante tener en cuenta sus posibles desventajas:

- Aptitud limitada para proyectos grandes: es posible que GitHub Flow no sea tan adecuado para proyectos a gran escala con bases de código complejas y múltiples ramas de funciones a largo plazo. En esos casos, un flujo de trabajo más estructurado, como Gitflow, podría proporcionar un mejor control sobre la gestión simultánea del desarrollo y las versiones.
- Falta de una estructura de publicación formal: GitHub Flow no define explícitamente un proceso de publicación ni admite funciones como el control de versiones, las revisiones o las ramas de mantenimiento. Esto puede ser una limitación para los proyectos que requieren una gestión estricta de las versiones o que necesitan soporte y mantenimiento a largo plazo.
- Soporte limitado para la planificación del lanzamiento a largo plazo: GitHub Flow se centra en las ramas de funciones de corta duración, que pueden no adaptarse bien a los proyectos que requieren una planificación del lanzamiento a largo plazo, como aquellos con hojas de ruta estrictas o amplias dependencias de funciones. Administrar calendarios de lanzamiento complejos puede resultar difícil debido a las limitaciones de Flow. GitHub
- Potencial de conflictos de fusión frecuentes: dado que GitHub Flow fomenta las ramificaciones y fusiones frecuentes, existe la posibilidad de que surjan conflictos de fusión, especialmente en

proyectos con mucha actividad de desarrollo. Resolver estos conflictos puede llevar mucho tiempo y puede requerir un esfuerzo adicional por parte del equipo de desarrollo.

- Falta de fases de flujo de trabajo formalizadas: GitHub Flow no define fases explícitas para el desarrollo, como las fases alfa, beta o candidata a una versión. Esto puede dificultar la comunicación del estado actual del proyecto o del nivel de estabilidad de las distintas ramas o versiones.
- Impacto de los cambios importantes: dado que GitHub Flow fomenta la fusión frecuente de los cambios en la `main` rama, existe un mayor riesgo de introducir cambios importantes que afecten a la estabilidad del código base. Las prácticas estrictas de revisión y prueba del código son cruciales para mitigar este riesgo de manera efectiva.

Estrategia de ramificación de Gitflow

Gitflow es un modelo de ramificación que implica el uso de múltiples ramas para mover el código del desarrollo a la producción. Gitflow funciona bien para los equipos que tienen ciclos de lanzamiento programados y que necesitan definir un conjunto de funciones como versión. El desarrollo se completa en ramas de funciones individuales que se fusionan, previa aprobación, en una rama de desarrollo, que se utiliza para la integración. Las funciones de esta rama se consideran listas para su producción. Cuando todas las funciones planificadas se han acumulado en la rama de desarrollo, se crea una rama de lanzamiento para las implementaciones en los entornos superiores. Esta separación mejora el control sobre qué cambios se trasladan a cada entorno determinado según un cronograma definido. Si es necesario, puede acelerar este proceso y convertirlo en un modelo de implementación más rápido.

Para obtener más información sobre la estrategia de ramificación de Gitflow, consulta los siguientes recursos:

- Implementa una estrategia de ramificación de Gitflow para entornos con varias cuentas DevOps (guía prescriptiva)AWS
- The original Gitflow blog (entrada en el blog de Vincent Driessen)
- Gitflow workflow (Atlassian)

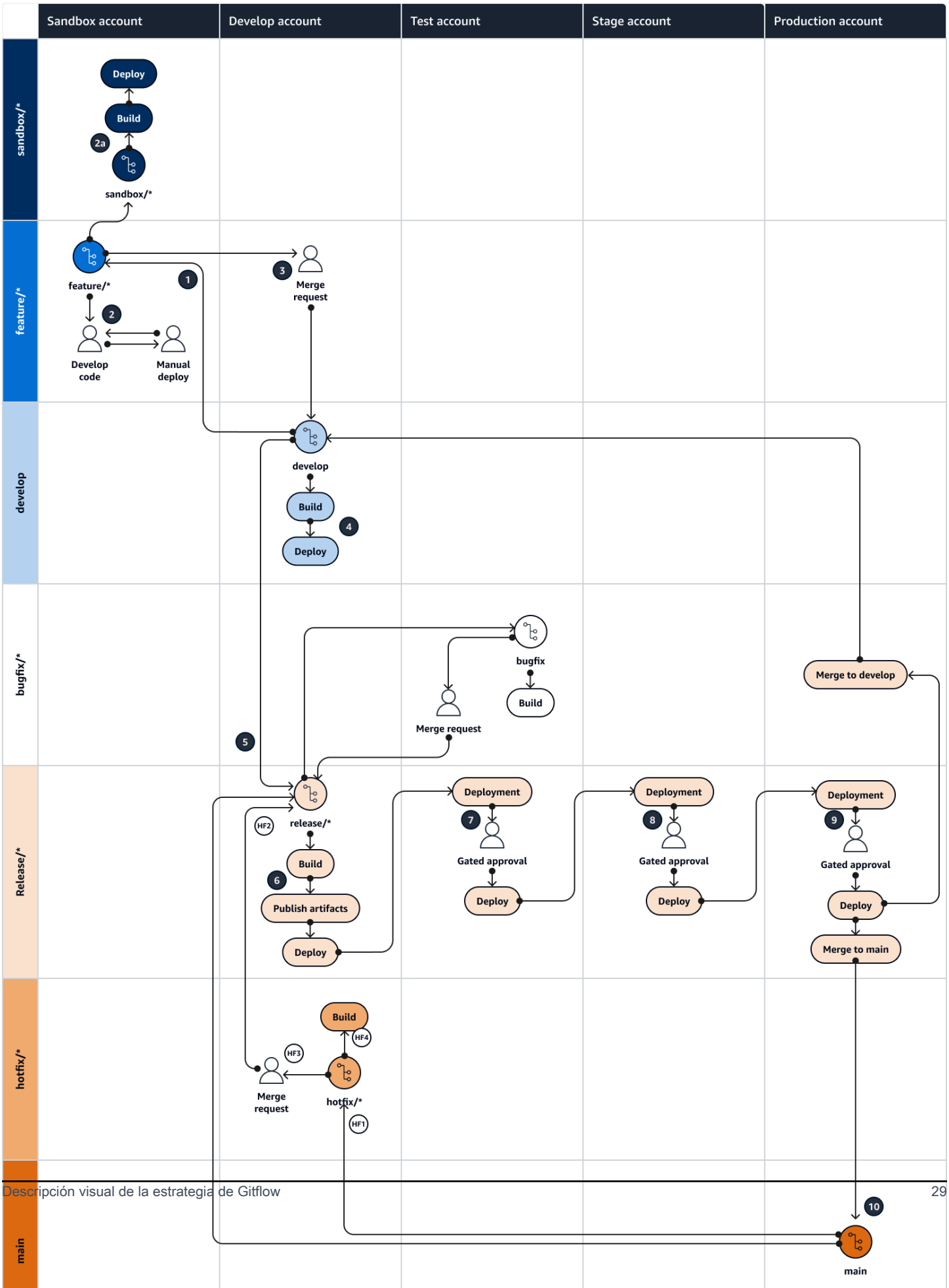
Temas de esta sección:

- Descripción visual de la estrategia de Gitflow
- Las ramificaciones en una estrategia de Gitflow

- Ventajas y desventajas de la estrategia de Gitflow

Descripción visual de la estrategia de Gitflow

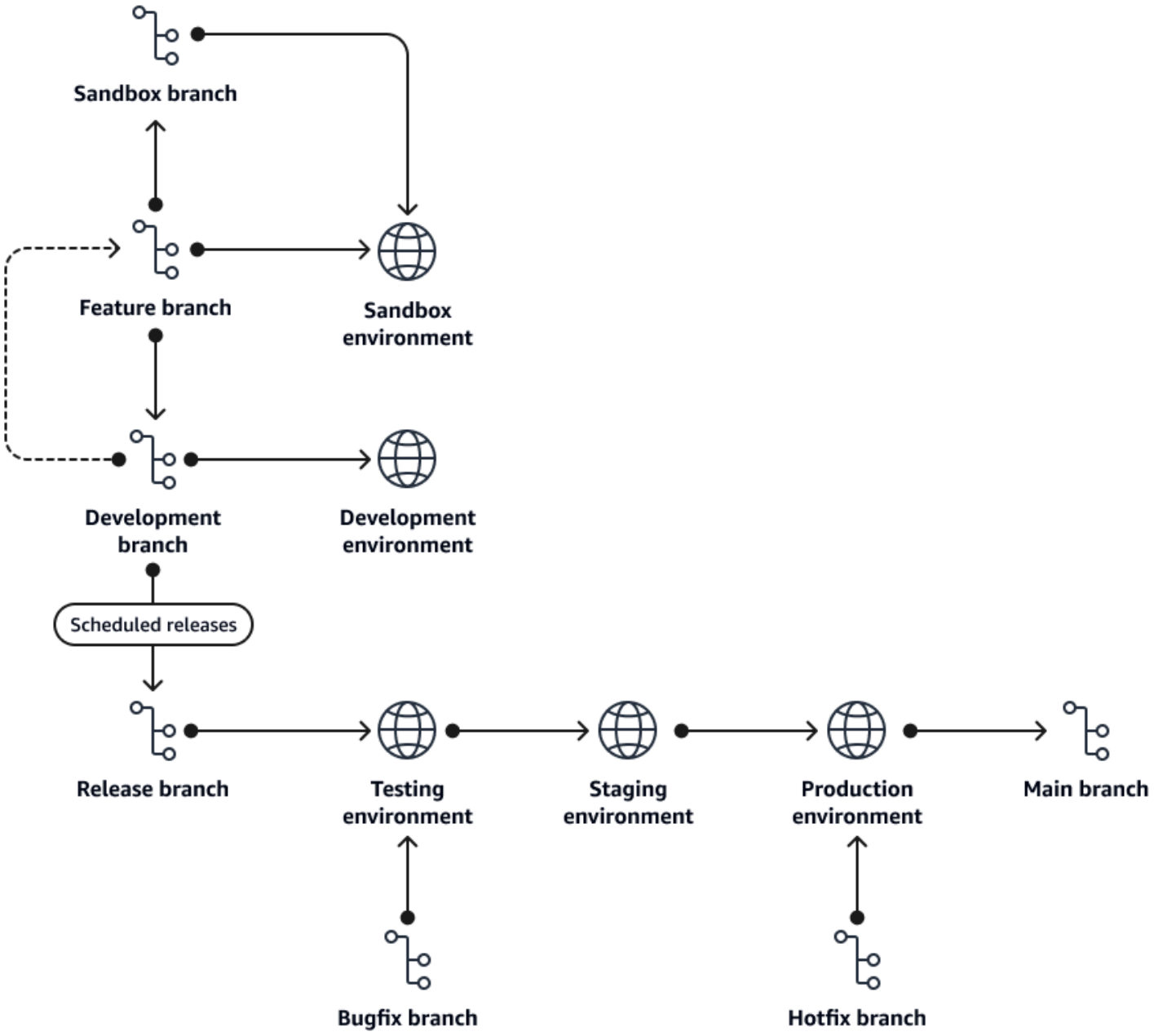
El siguiente diagrama puede usarse como un cuadro de Punnett para entender la estrategia de ramificación de Gitflow. Alinee las ramas del eje vertical con los AWS entornos del eje horizontal para determinar qué acciones realizar en cada escenario. Los números encerrados en un círculo te guían por la secuencia de acciones representada en el diagrama. Para obtener más información sobre las actividades que se llevan a cabo en cada entorno, consulte DevOps los entornos en esta guía.



Descripción visual de la estrategia de Gitflow

Las ramificaciones en una estrategia de Gitflow

Una estrategia de ramificación de Gitflow suele tener las siguientes ramas.



rama de característica

`Feature`las sucursales son ramas a corto plazo en las que se desarrollan funciones. La `feature` rama se crea al ramificarse a partir de la `develop` rama. Los desarrolladores iteran, confirman

y prueban el código de la `feature` rama. Cuando la función está completa, el desarrollador la promociona. Solo hay dos caminos hacia adelante desde una rama de funciones:

- Incorpórese a la `sandbox` rama
- Crea una solicitud de fusión en la `develop` sucursal

Convención de nomenclatura:	`feature/<story number>_<developer initials>_<descriptor>`
Ejemplo de convención de nomenclatura:	`feature/123456_MS_Implement_Feature_A`

rama sandbox

La `sandbox` rama es una rama no estándar y de corto plazo para Gitflow. Sin embargo, es útil para el desarrollo de CI/CD canalizaciones. La `sandbox` rama se utiliza principalmente para los siguientes fines:

- Realice una implementación completa en el entorno sandbox mediante las CI/CD canalizaciones en lugar de una implementación manual.
- Desarrolle y pruebe una canalización antes de enviar solicitudes de fusión para realizar pruebas completas en un entorno inferior, como el de desarrollo o las pruebas.

`Sandbox`las sucursales son de naturaleza temporal y no están destinadas a ser duraderas. Deben borrarse una vez finalizadas las pruebas específicas.

Convención de nomenclatura:	`sandbox/<story number>_<developer initials>_<descriptor>`
Ejemplo de convención de nomenclatura:	`sandbox/123456_MS_Test_Pipeline_Deploy`

desarrollar una rama

La `develop` sucursal es una rama de larga duración en la que las funciones se integran, crean, validan e implementan en el entorno de desarrollo. Todas las `feature` sucursales se fusionan en la `develop` sucursal. Las fusiones en la `develop` sucursal se realizan mediante una solicitud de fusión que requiere una compilación correcta y la aprobación de dos desarrolladores. Para evitar que se eliminen, habilita la protección de sucursales en la `develop` rama.

Convención de nomenclatura:	`develop`

rama de lanzamiento

En Gitflow, las `release` ramas son ramas a corto plazo. Estas ramas son especiales porque puedes desplegarlas en múltiples entornos, adoptando la metodología de construir una vez y desplegar muchas veces. `Release`las sucursales pueden centrarse en los entornos de prueba, puesta en escena o producción. Una vez que un equipo de desarrollo ha decidido promover funciones en entornos superiores, crea una nueva `release` rama y utiliza un aumento del número de versión con respecto a la versión anterior. En las puertas de cada entorno, las implementaciones requieren aprobaciones manuales para poder llevarse a cabo. `Release`las sucursales deberían requerir que se modifique una solicitud de fusión.

Una vez que la `release` rama se haya implementado en producción, se debe volver a fusionar con las `main` ramas `develop` y para garantizar que las correcciones de errores o revisiones se fusionen de nuevo en futuras iniciativas de desarrollo.

Convención de nomenclatura:	`release/v{major}.{minor}`
Ejemplo de convención de nomenclatura:	`release/v1.0`

rama principal

La `main` rama es una rama de larga duración que siempre representa el código que se está ejecutando en producción. El código se fusiona automáticamente en la `main` rama desde una rama de publicación tras una implementación correcta desde la canalización de versiones. Para evitar la eliminación, habilita la protección de la rama en la `main` rama.

Convención de nomenclatura:	`main`

rama de corrección de errores

La `bugfix` rama es una rama a corto plazo que se utiliza para solucionar problemas en las ramas de lanzamiento que no se han lanzado al mercado de producción. Una `bugfix` rama solo debe usarse para promover correcciones en las `release` sucursales para los entornos de prueba, preparación o producción. Una `bugfix` sucursal siempre se ramifica a partir de una `release` sucursal.

Una vez que se haya probado la corrección del error, se puede ascender a la `release` rama mediante una solicitud de fusión. A continuación, puedes impulsar la `release` rama siguiendo el proceso de publicación estándar.

Convención de nomenclatura:	`bugfix/<ticket>_<developer initials>_<descriptor>`
Ejemplo de convención de nomenclatura:	`bugfix/123456_MS_Fix_Problem_A`

rama de hotfix

La `hotfix` sucursal es una rama a corto plazo que se utiliza para solucionar problemas en la producción. Solo se utilizará para promover soluciones que deben acelerarse para que lleguen al entorno de producción. Siempre se `hotfix` ramifica una sucursal desde. `main`

Una vez que se haya probado la revisión, puedes pasarla a producción mediante una solicitud de fusión en la `release` rama desde la que se creó. `main` Para realizar las pruebas, puedes impulsar la `release` rama siguiendo el proceso de publicación estándar.

Convención de nomenclatura:	`hotfix/<ticket>_<developer initials>_<descriptor>`
Ejemplo de convención de nomenclatura:	`hotfix/123456_MS_Fix_Problem_A`

Ventajas y desventajas de la estrategia de Gitflow

La estrategia de ramificación de Gitflow se adapta bien a equipos más grandes y distribuidos que tienen requisitos estrictos de publicación y cumplimiento. Gitflow contribuye a un ciclo de lanzamiento predecible para la organización, algo que suelen preferir las organizaciones más grandes. Gitflow también es ideal para equipos que necesitan barreras para completar su ciclo de vida de desarrollo de software de forma adecuada. Esto se debe a que la estrategia incorpora múltiples oportunidades de revisión y control de calidad. Gitflow también es ideal para equipos que deben mantener varias versiones de versiones de producción simultáneamente. Algunas desventajas de GitFlow son que es más complejo que otros modelos de ramificación y requiere un estricto cumplimiento del patrón para completarse con éxito. Gitflow no funciona bien para las organizaciones que buscan una entrega continua debido a la naturaleza rígida de la gestión de las ramas de lanzamiento. Las ramas de publicación de Gitflow pueden ser ramas duraderas que pueden acumular deudas técnicas si no se gestionan adecuadamente y de manera oportuna.

Ventajas

Gitflow-based el desarrollo ofrece varias ventajas que pueden mejorar el proceso de desarrollo, agilizar la colaboración y mejorar la calidad general del software. Las siguientes son algunas de las principales ventajas:

- Proceso de publicación predecible: Gitflow sigue un proceso de publicación regular y predecible. Es ideal para equipos con cadencias de desarrollo y lanzamiento regulares.
- Colaboración mejorada: Gitflow fomenta el uso de sucursales `feature` y `release` sucursales. Estas dos ramas ayudan a los equipos a trabajar en paralelo con una dependencia mínima entre sí.
- Ideal para múltiples entornos: Gitflow usa `release` ramas, que pueden ser ramas de mayor duración. Estas sucursales permiten a los equipos centrarse en los lanzamientos individuales durante un período de tiempo más largo.
- Varias versiones en producción: si tu equipo admite varias versiones del software en producción, las `release` sucursales de Gitflow admiten este requisito.
- Built-in revisiones de la calidad del código: Gitflow exige y fomenta el uso de revisiones y aprobaciones del código antes de promocionarlo a otro entorno. Este proceso elimina las fricciones entre los desarrolladores al requerir este paso para todas las promociones de código.
- Beneficios organizativos: Gitflow también tiene ventajas a nivel organizativo. Gitflow fomenta el uso de un ciclo de lanzamiento estándar, lo que ayuda a la organización a entender y anticipar

el calendario de lanzamientos. Como la empresa ahora sabe cuándo se pueden ofrecer nuevas funciones, se reducen las fricciones con respecto a los plazos, ya que hay fechas de entrega establecidas.

Desventajas

Gitflow-based el desarrollo tiene algunas desventajas que pueden afectar al proceso de desarrollo y a la dinámica del equipo. Los siguientes son algunos inconvenientes notables:

- Complejidad: Gitflow es un patrón complejo que deben aprender los nuevos equipos, y debes seguir las reglas de Gitflow para usarlo con éxito.
- Despliegue continuo: Gitflow no se ajusta a un modelo en el que muchas implementaciones se lanzan a producción de forma rápida. Esto se debe a que Gitflow requiere el uso de varias sucursales y un flujo de trabajo estricto que regule la sucursal. `release`
- Administración de sucursales: Gitflow utiliza muchas sucursales, lo que puede resultar engorroso de mantener. Puede resultar difícil rastrear las distintas ramas y fusionar el código publicado para mantener las ramas correctamente alineadas entre sí.
- Deuda técnica: dado que las versiones de Gitflow suelen ser más lentas que los otros modelos de ramificación, es posible que se acumulen más funciones antes de publicarlas, lo que puede provocar que se acumule deuda técnica.

Los equipos deben considerar detenidamente estos inconvenientes a la hora de decidir si Gitflow-based el desarrollo es el enfoque adecuado para su proyecto.

Siguientes pasos

Esta guía explica las diferencias entre tres estrategias comunes de ramificación de Git: GitHub Flow, Gitflow y Trunk. Describe sus flujos de trabajo en detalle y también proporciona las ventajas y desventajas de cada uno de ellos. Los siguientes pasos son elegir uno de estos flujos de trabajo estándar para su organización. Para implementar una de estas estrategias de ramificación, consulta lo siguiente:

- Implemente una estrategia de ramificación troncal para entornos con varias cuentas DevOps
- Implemente una estrategia de ramificación GitHub de Flow para entornos con varias cuentas DevOps
- Implementa una estrategia de ramificación de Gitflow para entornos con varias cuentas DevOps

Si no estás seguro de por dónde empezar el viaje de tu equipo hacia el uso de Git y DevOps los procesos, te recomendamos que elijas una solución estándar y la pruebes. El uso de una convención de ramificación estándar ayuda al equipo a aprovechar la documentación existente y a aprender qué es lo que funciona mejor para ellos.

No dudes en cambiar tu estrategia si no funciona para tu organización o tus equipos de desarrollo. Las necesidades y los requisitos de los equipos de desarrollo pueden cambiar con el tiempo y no existe una solución única y perfecta.

Recursos

Esta guía no incluye formación sobre Git; sin embargo, hay muchos recursos de alta calidad disponibles en Internet si necesitas esta formación. Te recomendamos que comiences por el sitio de documentación de Git.

Los siguientes recursos pueden ayudarte con tu viaje de ramificación de Git en el Nube de AWS.

AWS Guía prescriptiva

- Implemente una estrategia de ramificación troncal para entornos de cuentas múltiples DevOps
- Implemente una estrategia de ramificación GitHub de Flow para entornos con varias cuentas DevOps
- Implementa una estrategia de ramificación de Gitflow para entornos con varias cuentas DevOps

Otra orientación AWS

- AWS DevOps Orientación
- AWS Arquitectura de referencia para la canalización de despliegue
- ¿Qué es DevOps?
- DevOpsrecursos

Otros recursos

- Metodología de aplicaciones de doce factores (12factor.net)
- Secretos de Git () GitHub
- Gitflow
  - El blog original de Gitflow (entrada del blog de Vincent Driessen)
  - Flujo de trabajo de Gitflow (Atlassian)
  - Gitflow en GitHub: Cómo usar los flujos de trabajo de Git Flow con repositorios GitHub basados (vídeo) YouTube
  - Ejemplo de Git Flow Init (YouTube vídeo)

  - La rama de lanzamiento de Gitflow de principio a fin (vídeo) YouTube
- GitHub Flujo
  - GitHub Flow Quickstart (GitHub documentación)
  - ¿Por qué GitHub Flow? (Sitio web GitHub de Flow)
- Tronco
  - Introducción al desarrollo basado en troncos (sitio web de desarrollo basado en troncos)

Colaboradores

Creación

- Mike Stephens, arquitecto sénior de aplicaciones en la nube, AWS
- Rayjan Wilson, arquitecto sénior de aplicaciones en la nube, AWS
- Abhilash Vinod, jefe de equipo, arquitecto sénior de aplicaciones en la nube, AWS

Revisando

- Stephen DiCato, consultor sénior de seguridad, AWS
- Gaurav Samudra, arquitecto de aplicaciones en la nube, AWS
- Steven Guggenheimer, jefe de equipo, arquitecto sénior de aplicaciones en la nube, AWS

Redacción técnica

- Lilly AbouHarb, redactora técnica sénior, AWS

Historial de revisión

En la siguiente tabla, se describen cambios significativos de esta guía. Si quiere recibir notificaciones de futuras actualizaciones, puede suscribirse a las notificaciones RSS.

Cambio	Descripción	Fecha
Actualización	Sustituimos las imágenes en las secciones Branches en una estrategia Trunk, Branches en una estrategia GitHub Flow y Branches en una estrategia Gitflow.	5 de junio de 2026
Publicación inicial	—	15 de febrero de 2024

Las traducciones son generadas a través de traducción automática. En caso de conflicto entre la traducción y la version original de inglés, prevalecerá la version en inglés.