Memilih strategi percabangan Git untuk lingkungan multi-akun DevOps

# AWS Panduan Preskriptif

# AWS Panduan Preskriptif: Memilih strategi percabangan Git untuk lingkungan multi-akun DevOps

# Table of Contents

# Memilih strategi percabangan Git untuk lingkungan multi-akun DevOps

Amazon Web Services (kontributor)

Februari 2024 (riwayat dokumen)

Pindah ke pendekatan berbasis cloud dan memberikan solusi perangkat lunak AWS dapat bersifat transformatif. Ini mungkin memerlukan perubahan pada proses siklus hidup pengembangan perangkat lunak Anda. Biasanya, beberapa Akun AWS digunakan selama proses pengembangan di AWS Cloud. Memilih strategi percabangan Git yang kompatibel untuk dipasangkan dengan DevOps proses Anda sangat penting untuk kesuksesan. Memilih strategi percabangan Git yang tepat untuk organisasi Anda membantu Anda mengkomunikasikan DevOps standar dan praktik terbaik secara ringkas di seluruh tim pengembangan. Percabangan Git dapat sederhana dalam satu lingkungan, tetapi dapat menjadi membingungkan ketika diterapkan di beberapa lingkungan, seperti kotak pasir, pengembangan, pengujian, pementasan, dan lingkungan produksi. Memiliki beberapa lingkungan meningkatkan kompleksitas DevOps implementasi.

Panduan ini menyediakan diagram visual strategi percabangan Git yang menunjukkan bagaimana organisasi dapat menerapkan proses multi-akun. DevOps Panduan visual membantu tim memahami cara menggabungkan strategi percabangan Git mereka dengan praktik mereka DevOps . Menggunakan model percabangan standar, seperti GitHub Gitflow, Flow, atau Trunk, untuk mengelola repositori kode sumber membantu tim pengembangan menyelaraskan pekerjaan mereka. Tim-tim ini juga dapat menggunakan sumber daya pelatihan Git standar di internet untuk memahami dan menerapkan model dan strategi tersebut.

Untuk praktik DevOps terbaik AWS, tinjau DevOpsPanduan dalam AWS Well-Architected. Saat Anda meninjau panduan ini, gunakan uji tuntas untuk memilih strategi percabangan yang tepat untuk organisasi Anda. Beberapa strategi mungkin cocok dengan kasus penggunaan Anda lebih baik daripada yang lain.

## Tujuan

Panduan ini adalah bagian dari seri dokumentasi tentang memilih dan menerapkan strategi DevOps percabangan untuk organisasi dengan banyak Akun AWS. Seri ini dirancang untuk membantu Anda menerapkan strategi yang paling memenuhi persyaratan, sasaran, dan praktik terbaik Anda sejak

awal, untuk merampingkan pengalaman Anda di dalamnya. AWS Cloud Panduan ini tidak berisi skrip yang DevOps dapat dieksekusi karena mereka bervariasi berdasarkan kerangka kerja mesin dan teknologi continuous integration and continuous delivery (CI/CD) yang digunakan organisasi Anda.

Panduan ini menjelaskan perbedaan antara tiga strategi percabangan Git yang umum: GitHub Flow, Gitflow, dan Trunk. Rekomendasi dalam panduan ini membantu tim mengidentifikasi strategi percabangan yang selaras dengan tujuan organisasi mereka. Setelah meninjau panduan ini, Anda harus dapat memilih strategi percabangan untuk organisasi Anda. Setelah memilih strategi, Anda dapat menggunakan salah satu pola berikut untuk membantu Anda menerapkan strategi itu dengan tim pengembangan Anda:

- Menerapkan strategi percabangan Trunk untuk lingkungan multi-akun DevOps
- Menerapkan strategi percabangan GitHub Flow untuk lingkungan multi-akun DevOps
- Menerapkan strategi percabangan Gitflow untuk lingkungan multi-akun DevOps

Penting untuk dicatat bahwa apa yang berhasil untuk satu organisasi, tim, atau proyek mungkin tidak cocok untuk orang lain. Pilihan antara strategi percabangan Git tergantung pada berbagai faktor, seperti ukuran tim, persyaratan proyek, dan keseimbangan yang diinginkan antara kolaborasi, frekuensi integrasi, dan manajemen rilis.

## Menggunakan CI/CD praktik

AWS merekomendasikan agar Anda menerapkan integrasi berkelanjutan dan pengiriman berkelanjutan (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/CDjaringan pipa juga menyediakan tata kelola dan pagar pembatas untuk tim pengembangan. Mereka menegakkan konsistensi, standar, praktik terbaik, dan tingkat penerimaan minimum untuk penerimaan dan penerapan fitur. Untuk informasi selengkapnya, lihat Mempraktikkan Integrasi Berkelanjutan dan Pengiriman Berkelanjutan di AWS.

Semua strategi percabangan yang dibahas dalam panduan ini sangat cocok untuk 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 menetapkan pagar pembatas dan mekanisme untuk tim pengembangan yang mencegah pengembang dari sengaja atau tidak sengaja mengelilingi proses yang ditentukan.

AWS menawarkan serangkaian layanan pengembang yang dirancang untuk membantu Anda membangun pipa CI/CD. Misalnya, AWS CodePipelineadalah layanan pengiriman berkelanjutan yang dikelola sepenuhnya yang membantu Anda mengotomatiskan saluran pipa rilis untuk pembaruan aplikasi dan infrastruktur yang cepat dan andal. AWS CodeBuildmengkompilasi kode sumber, menjalankan tes, dan menghasilkan paket ready-to-deploy perangkat lunak. Untuk informasi selengkapnya, lihat Alat Pengembang di AWS.

# Memahami DevOps lingkungan

Untuk memahami strategi percabangan, Anda harus memahami tujuan dan kegiatan yang terjadi di setiap lingkungan. Membangun beberapa lingkungan membantu Anda memisahkan aktivitas pengembangan menjadi beberapa tahap, memantau aktivitas tersebut, dan mencegah pelepasan fitur yang tidak disetujui secara tidak disengaja. Anda dapat memiliki satu atau lebih Akun AWS di setiap lingkungan.

Sebagian besar organisasi memiliki beberapa lingkungan yang diuraikan untuk digunakan. Namun, jumlah lingkungan dapat bervariasi menurut organisasi dan sesuai dengan kebijakan pengembangan perangkat lunak. Seri dokumentasi ini mengasumsikan bahwa Anda memiliki lima lingkungan umum berikut yang menjangkau pipeline pengembangan Anda, meskipun mereka mungkin dipanggil dengan nama yang berbeda:

- Sandbox — Lingkungan di mana pengembang menulis kode, membuat kesalahan, dan melakukan bukti kerja konsep.
- Pengembangan — Lingkungan di mana pengembang mengintegrasikan kode mereka untuk mengonfirmasi bahwa semuanya berfungsi sebagai aplikasi tunggal yang kohesif.
- Pengujian — Lingkungan di mana tim QA atau pengujian penerimaan berlangsung. Tim sering melakukan pengujian kinerja atau integrasi di lingkungan ini.
- Staging — Lingkungan praproduksi tempat Anda memvalidasi bahwa kode dan infrastruktur berfungsi seperti yang diharapkan dalam keadaan setara produksi. Lingkungan ini dikonfigurasi agar semirip mungkin dengan lingkungan produksi.
- Produksi — Lingkungan yang menangani lalu lintas dari pengguna akhir dan pelanggan Anda.

Bagian ini menjelaskan setiap lingkungan secara rinci. Ini juga menjelaskan langkah-langkah build, langkah penerapan, dan kriteria keluar untuk setiap lingkungan sehingga Anda dapat melanjutkan ke yang berikutnya. Gambar berikut menunjukkan lingkungan ini secara berurutan.


Topik di bagian ini:

# Lingkungan kotak pasir

Lingkungan kotak pasir adalah tempat pengembang menulis kode, membuat kesalahan, dan melakukan bukti kerja konsep. Anda dapat menerapkan ke lingkungan sandbox dari workstation lokal atau melalui skrip di workstation lokal.

## Akses

Pengembang harus memiliki akses penuh ke lingkungan kotak pasir.

## Membangun langkah

Pengembang menjalankan build secara manual di workstation lokal mereka saat mereka siap menerapkan perubahan ke lingkungan kotak pasir.

1. Gunakan git-secrets (GitHub) untuk memindai informasi sensitif
2. Lint kode sumber
3. Membangun dan mengkompilasi kode sumber, jika berlaku
4. Lakukan pengujian unit
5. Lakukan analisis cakupan kode
6. Lakukan analisis kode statis
7. Membangun infrastruktur sebagai kode (IAc)
8. Lakukan analisis keamanan IAc
9. Ekstrak lisensi open source
10. Publikasikan artefak build

## Langkah-langkah penyebaran

Jika Anda menggunakan model Gitflow atau Trunk, langkah penerapan secara otomatis dimulai ketika `feature` cabang berhasil dibangun di lingkungan kotak pasir. Jika Anda menggunakan model GitHub Flow, Anda akan melakukan langkah penerapan berikut secara manual. Berikut ini adalah langkah-langkah penerapan di lingkungan sandbox:

1. Unduh artefak yang diterbitkan
2. Lakukan pembuatan versi database
3. Lakukan penyebaran IAc
4. Lakukan pengujian integrasi

## Harapan sebelum pindah ke lingkungan pembangunan

- Sukses membangun `feature` cabang di lingkungan kotak pasir
- Pengembang telah menerapkan dan menguji fitur secara manual di lingkungan kotak pasir

# Lingkungan pengembangan

Lingkungan pengembangan adalah tempat pengembang mengintegrasikan kode mereka bersama-sama untuk memastikan semuanya berfungsi sebagai satu aplikasi yang kohesif. Di Gitflow, lingkungan pengembangan berisi fitur terbaru yang disertakan oleh permintaan gabungan dan siap untuk dirilis. Dalam strategi GitHub Flow and Trunk, lingkungan pengembangan dianggap sebagai lingkungan pengujian, dan basis kode mungkin tidak stabil dan tidak cocok untuk diterapkan ke produksi.

## Akses

Tetapkan izin sesuai dengan prinsip hak istimewa paling sedikit. Keistimewaan paling sedikit adalah praktik keamanan terbaik untuk memberikan izin minimum yang diperlukan untuk melakukan tugas. Pengembang harus memiliki lebih sedikit akses ke lingkungan pengembangan daripada yang mereka miliki ke lingkungan kotak pasir.

## Membangun langkah

Membuat permintaan gabungan ke `develop` cabang (Gitflow) atau `main` cabang (Trunk atau GitHub Flow) secara otomatis memulai build.

1. Gunakan git-secrets (GitHub) untuk memindai informasi sensitif
2. Lint kode sumber
3. Membangun dan mengkompilasi kode sumber, jika berlaku
4. Lakukan pengujian unit
5. Lakukan analisis cakupan kode
6. Lakukan analisis kode statis
7. Membangun IAc
8. Lakukan analisis keamanan IAc
9. Ekstrak lisensi open source

## Langkah-langkah penyebaran

Jika Anda menggunakan model Gitflow, langkah penerapan secara otomatis dimulai ketika `develop` cabang berhasil dibangun di lingkungan pengembangan. Jika Anda menggunakan model GitHub Flow atau model Trunk, maka langkah penerapan akan dimulai secara otomatis saat permintaan gabungan dibuat terhadap cabang. `main` Berikut ini adalah langkah-langkah penerapan di lingkungan pengembangan:

1. Unduh artefak yang diterbitkan dari langkah-langkah pembuatan
2. Lakukan pembuatan versi database
3. Lakukan penyebaran IAc
4. Lakukan tes integrasi

## Harapan sebelum pindah ke lingkungan pengujian

- Pembuatan dan penerapan `develop` cabang (Gitflow) atau `main` cabang (Trunk atau GitHub Flow) yang berhasil di lingkungan pengembangan
- Pengujian unit lolos pada 100%

- Membangun IAc yang sukses
- Artefak penyebaran berhasil dibuat
- Pengembang telah melakukan verifikasi manual untuk mengonfirmasi bahwa fitur tersebut berfungsi seperti yang diharapkan

# Lingkungan pengujian

Personel jaminan kualitas (QA) menggunakan lingkungan pengujian untuk memvalidasi fitur. Mereka menyetujui perubahan setelah mereka menyelesaikan pengujian. Ketika mereka menyetujui, cabang pindah ke lingkungan berikutnya, pementasan. Di Gitflow, lingkungan ini dan lainnya di atasnya hanya tersedia untuk penerapan dari `release` cabang. `release`Cabang didasarkan pada `develop` cabang yang berisi fitur yang direncanakan.

## Akses

Tetapkan izin sesuai dengan prinsip hak istimewa paling sedikit. Pengembang harus memiliki lebih sedikit akses ke lingkungan pengujian daripada yang mereka miliki ke lingkungan pengembangan. Personel QA memerlukan izin yang cukup untuk menguji fitur.

## Membangun langkah

Proses pembuatan di lingkungan ini hanya berlaku untuk perbaikan bug saat menggunakan strategi Gitflow. Membuat permintaan gabungan ke `bugfix` cabang secara otomatis memulai pembuatan.

1. Gunakan git-secrets (GitHub) untuk memindai informasi sensitif
2. Lint kode sumber
3. Membangun dan mengkompilasi kode sumber, jika berlaku
4. Lakukan pengujian unit
5. Lakukan analisis cakupan kode
6. Lakukan analisis kode statis
7. Membangun IAc
8. Lakukan analisis keamanan IAc
9. Ekstrak lisensi open source

## Langkah-langkah penyebaran

Secara otomatis memulai penerapan `release` cabang (Gitflow) atau `main` cabang (Trunk atau GitHub Flow) di lingkungan pengujian setelah penerapan di lingkungan pengembangan. Berikut ini adalah langkah-langkah penerapan di lingkungan pengujian:

1. Menyebarkan `release` cabang (Gitflow) atau `main` cabang (Trunk atau GitHub Flow) di lingkungan pengujian
2. Jeda untuk persetujuan manual oleh personel yang ditunjuk
3. Unduh artefak yang diterbitkan
4. Lakukan pembuatan versi database
5. Lakukan penyebaran IAc
6. Lakukan tes integrasi
7. Lakukan tes kinerja
8. Persetujuan jaminan kualitas

## Harapan sebelum pindah ke lingkungan pementasan

- Tim pengembangan dan QA telah melakukan pengujian yang cukup untuk memenuhi kebutuhan organisasi Anda.
- Tim pengembangan telah menyelesaikan bug yang ditemukan melalui `bugfix` cabang.

# Lingkungan pementasan

Lingkungan pementasan dikonfigurasi agar sama dengan lingkungan produksi. Misalnya, pengaturan data harus serupa dalam ruang lingkup dan ukuran dengan beban kerja produksi. Gunakan lingkungan pementasan untuk memverifikasi bahwa kode dan infrastruktur beroperasi seperti yang diharapkan. Lingkungan ini juga merupakan pilihan yang lebih disukai untuk kasus penggunaan bisnis, seperti pratinjau atau demonstrasi pelanggan.

## Akses

Tetapkan izin sesuai dengan prinsip hak istimewa paling sedikit. Pengembang harus memiliki akses yang sama ke lingkungan pementasan seperti yang mereka lakukan pada lingkungan produksi.

## Membangun langkah

Tidak ada. Artefak yang sama yang digunakan dalam lingkungan pengujian digunakan kembali di lingkungan pementasan.

## Langkah-langkah penyebaran

Secara otomatis memulai penerapan `release` cabang (Gitflow) atau `main` cabang (Trunk atau GitHub Flow) di lingkungan pementasan setelah persetujuan dan penerapan di lingkungan pengujian. Berikut ini adalah langkah-langkah penerapan di lingkungan pementasan:

1. Menyebarkan `release` cabang (Gitflow) atau `main` cabang (Trunk atau GitHub Flow) di lingkungan pementasan
2. Jeda untuk persetujuan manual oleh personel yang ditunjuk
3. Unduh artefak yang diterbitkan
4. Lakukan pembuatan versi database
5. Lakukan penyebaran IAc
6. (Opsional) Lakukan pengujian integrasi
7. (Opsional) Lakukan pengujian beban
8. Dapatkan persetujuan dari pemberi persetujuan pengembangan, QA, produk, atau bisnis yang diperlukan

## Harapan sebelum pindah ke lingkungan produksi

- Rilis setara produksi telah berhasil diterapkan ke lingkungan pementasan
- (Opsional) Integrasi dan pengujian beban berhasil

# Lingkungan produksi

Lingkungan produksi mendukung produk yang dirilis, menangani data nyata oleh klien nyata. Ini adalah lingkungan yang dilindungi yang diberi akses dengan hak istimewa paling sedikit dan akses yang ditinggikan hanya boleh diizinkan melalui proses pengecualian yang diaudit untuk jangka waktu terbatas.

## Akses

Di lingkungan produksi, pengembang harus memiliki akses terbatas dan hanya-baca di AWS Management Console. Misalnya, pengembang harus dapat mengakses data log untuk operasi sehari-hari. Semua rilis ke produksi harus dipastikan dengan langkah persetujuan sebelum penerapan.

## Membangun langkah

Tidak ada. Artefak yang sama yang digunakan dalam lingkungan pengujian dan pementasan digunakan kembali di lingkungan produksi.

## Langkah-langkah penyebaran

Secara otomatis memulai penerapan `release` cabang (Gitflow) atau `main` cabang (Trunk atau GitHub Flow) di lingkungan produksi setelah persetujuan dan penerapan di lingkungan pementasan. Berikut ini adalah langkah-langkah penerapan di lingkungan produksi:

1. Menyebarkan `release` cabang (Gitflow) atau `main` cabang (Trunk atau GitHub Flow) di lingkungan produksi
2. Jeda untuk persetujuan manual oleh personel yang ditunjuk
3. Unduh artefak yang diterbitkan
4. Lakukan pembuatan versi database
5. Lakukan penyebaran IAc

# Praktik terbaik untuk pengembangan berbasis Git

Agar berhasil mengadopsi pengembangan berbasis GIT, penting untuk mengikuti serangkaian praktik terbaik yang mempromosikan kolaborasi, menjaga kualitas kode, dan mendukung integrasi berkelanjutan dan pengiriman berkelanjutan (CI/CD). Selain praktik terbaik dalam panduan ini, tinjau Panduan AWS DevOps Well-Architected. Berikut ini adalah beberapa praktik terbaik utama untuk pengembangan berbasis GIT pada: AWS

- Pertahankan perubahan kecil dan sering — Dorong pengembang untuk melakukan perubahan atau fitur kecil dan bertahap. Ini mengurangi risiko konflik gabungan dan membuatnya lebih mudah untuk mengidentifikasi dan memperbaiki masalah dengan cepat.
- Gunakan sakelar fitur - Untuk mengelola rilis fitur yang tidak lengkap atau eksperimental, gunakan sakelar fitur atau tanda fitur. Ini membantu Anda menyembunyikan, mengaktifkan, atau menonaktifkan fitur tertentu dalam produksi tanpa memengaruhi stabilitas cabang utama.
- Pertahankan rangkaian pengujian yang kuat — Rangkaian pengujian yang komprehensif dan terawat dengan baik sangat penting untuk mendeteksi masalah lebih awal dan memverifikasi bahwa basis kode tetap stabil. Investasikan dalam otomatisasi pengujian dan prioritaskan memperbaiki tes yang gagal.
- Merangkul integrasi berkelanjutan - Gunakan alat dan praktik integrasi berkelanjutan untuk secara otomatis membangun, menguji, dan mengintegrasikan perubahan kode ke `develop` cabang (Gitflow) atau `main` cabang (Trunk atau GitHub Flow). Ini membantu Anda menangkap masalah lebih awal dan merampingkan proses pengembangan.
- Lakukan tinjauan kode — Dorong peer review kode untuk menjaga kualitas, berbagi pengetahuan, dan menangkap potensi masalah sebelum diintegrasikan ke dalam `main` cabang. Gunakan permintaan tarik atau alat peninjauan kode lainnya untuk memfasilitasi proses ini.
- Pantau dan perbaiki build yang rusak — Saat build rusak atau pengujian gagal, prioritaskan memperbaiki masalah sesegera mungkin. Ini membuat `develop` cabang (Gitflow) atau `main` cabang (Trunk atau GitHub Flow) dalam keadaan yang dapat dirilis dan meminimalkan dampaknya pada pengembang lain.
- Berkomunikasi dan berkolaborasi - Mempromosikan komunikasi terbuka dan kolaborasi di antara anggota tim. Pastikan pengembang mengetahui pekerjaan yang sedang berlangsung dan perubahan yang dilakukan pada basis kode.

- Refactor terus menerus - Memfaktorkan ulang basis kode secara teratur untuk meningkatkan pemeliharaan dan mengurangi utang teknis. Dorong pengembang untuk meninggalkan kode dalam keadaan yang lebih baik daripada yang mereka temukan.
- Gunakan cabang berumur pendek untuk tugas kompleks — Untuk tugas yang lebih besar atau lebih kompleks, gunakan cabang berumur pendek (juga dikenal sebagai cabang tugas) untuk mengerjakan perubahan. Namun, pastikan untuk menjaga umur cabang tetap pendek, biasanya kurang dari sehari. Gabungkan perubahan kembali ke `develop` cabang (Gitflow) atau `main` cabang (Trunk atau GitHub Flow) sesegera mungkin. Penggabungan dan ulasan yang lebih kecil dan lebih sering lebih mudah digunakan dan diproses oleh tim daripada satu permintaan penggabungan besar.
- Latih dan dukung tim — Memberikan pelatihan dan dukungan kepada pengembang yang baru mengenal pengembangan berbasis GIT atau yang membutuhkan panduan dalam mengadopsi praktik terbaiknya.

# Strategi percabangan Git

Dalam urutan yang paling tidak hingga paling kompleks, panduan ini menjelaskan strategi Git-based percabangan berikut secara rinci:

- Trunk — Trunk-based pengembangan adalah praktik pengembangan perangkat lunak di mana semua pengembang bekerja pada satu cabang, biasanya disebut `main` cabang `trunk` atau. Ide di balik pendekatan ini adalah untuk menjaga basis kode dalam keadaan yang dapat dirilis secara terus menerus dengan mengintegrasikan perubahan kode secara sering dan mengandalkan pengujian otomatis dan integrasi berkelanjutan.
- GitHub GitHub Flow — Flow adalah alur kerja ringan berbasis cabang yang dikembangkan oleh. GitHub Ini didasarkan pada gagasan `feature` cabang berumur pendek. Ketika fitur selesai dan siap untuk digunakan, fitur tersebut digabungkan ke dalam cabang. `main`
- Gitflow — Dengan pendekatan Gitflow, pengembangan diselesaikan di masing-masing cabang fitur. Setelah persetujuan, Anda menggabungkan `feature` cabang menjadi cabang integrasi yang biasanya diberi nama`develop`. Ketika cukup banyak fitur telah terakumulasi di `develop` `release` cabang, cabang dibuat untuk menyebarkan fitur ke lingkungan atas.

Setiap strategi percabangan memiliki kelebihan dan kekurangan. Meskipun mereka semua menggunakan lingkungan yang sama, mereka tidak semua menggunakan cabang yang sama atau langkah persetujuan manual. Di bagian panduan ini, tinjau setiap strategi percabangan secara rinci sehingga Anda terbiasa dengan nuansanya dan dapat mengevaluasi apakah itu sesuai dengan kasus penggunaan organisasi Anda.

Topik di bagian ini:

- Strategi percabangan batang
- GitHub Strategi percabangan aliran
- Strategi percabangan Gitflow

## Strategi percabangan batang

Trunk-based pengembangan adalah praktik pengembangan perangkat lunak di mana semua pengembang bekerja pada satu cabang, biasanya disebut `main` cabang `trunk` atau. Ide di balik pendekatan ini adalah untuk menjaga basis kode dalam keadaan yang dapat dirilis secara terus

menerus dengan mengintegrasikan perubahan kode secara sering dan mengandalkan pengujian otomatis dan integrasi berkelanjutan.

Dalam pengembangan berbasis batang, pengembang melakukan perubahan mereka ke `main` cabang beberapa kali sehari, bertujuan untuk pembaruan kecil dan bertahap. Ini memungkinkan loop umpan balik cepat, mengurangi risiko konflik gabungan, dan mendorong kolaborasi di antara anggota tim. Praktik ini menekankan pentingnya rangkaian pengujian yang terpelihara dengan baik karena mengandalkan pengujian otomatis untuk menangkap potensi masalah lebih awal dan memastikan bahwa basis kode tetap stabil dan dapat dirilis.

Trunk-based pengembangan sering dikontraskan dengan pengembangan berbasis fitur (juga dikenal sebagai percabangan fitur atau pengembangan berbasis fitur), di mana setiap fitur baru atau perbaikan bug dikembangkan di cabang khusus sendiri, terpisah dari cabang utama. Pilihan antara pengembangan berbasis batang dan pengembangan berbasis fitur tergantung pada faktor-faktor seperti ukuran tim, persyaratan proyek, dan keseimbangan yang diinginkan antara kolaborasi, frekuensi integrasi, dan manajemen rilis.

Untuk informasi lebih lanjut tentang strategi percabangan Trunk, lihat sumber daya berikut:

- Menerapkan strategi percabangan Trunk untuk DevOps lingkungan multi-akun (Panduan AWS Preskriptif)
- Pengantar Trunk-Based Pengembangan (situs web Pengembangan Berbasis Batang)

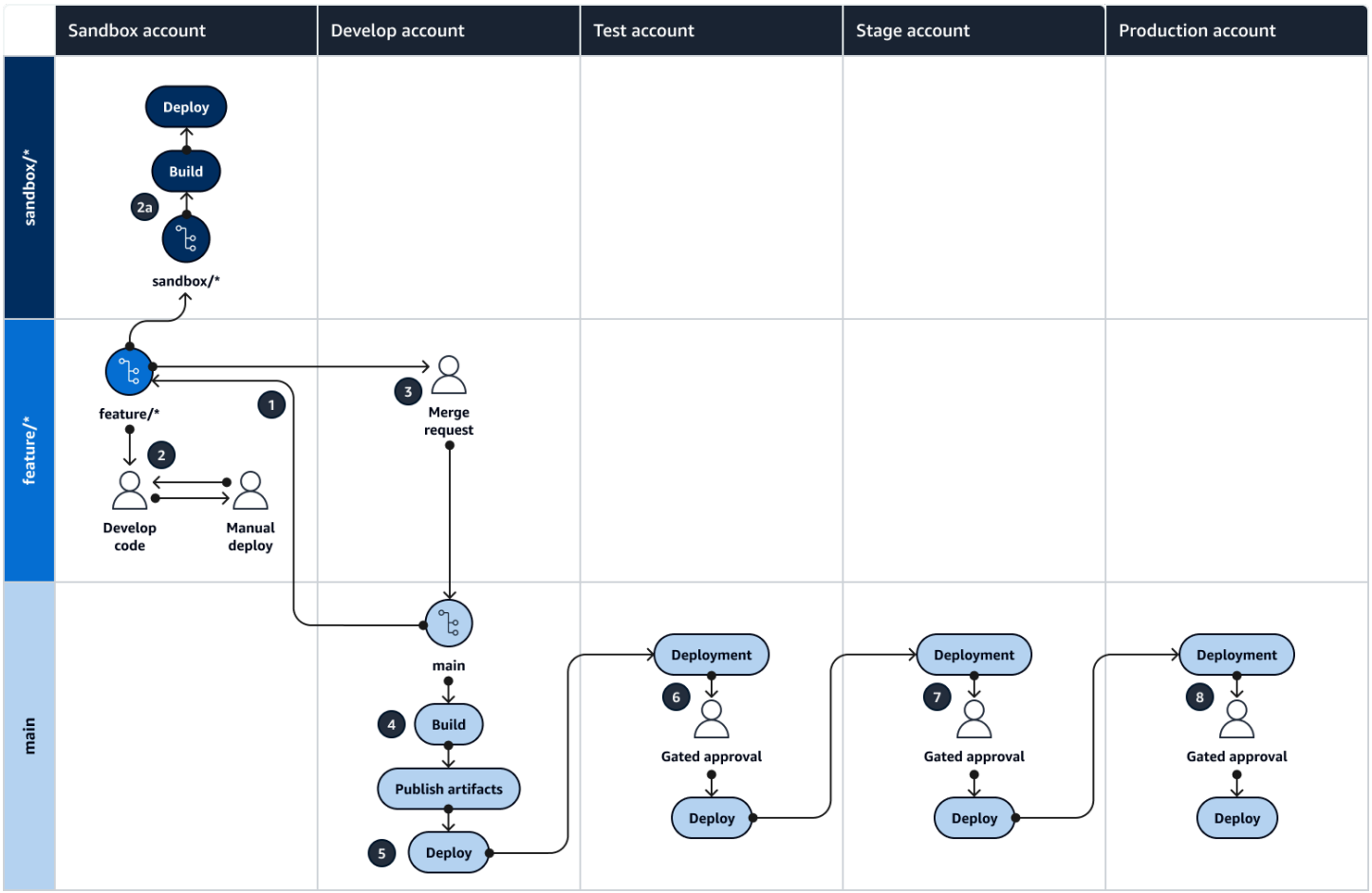
Topik di bagian ini:

- Gambaran visual dari strategi Trunk
- Cabang dalam strategi Trunk
- Keuntungan dan kerugian dari strategi Trunk

## Gambaran visual dari strategi Trunk

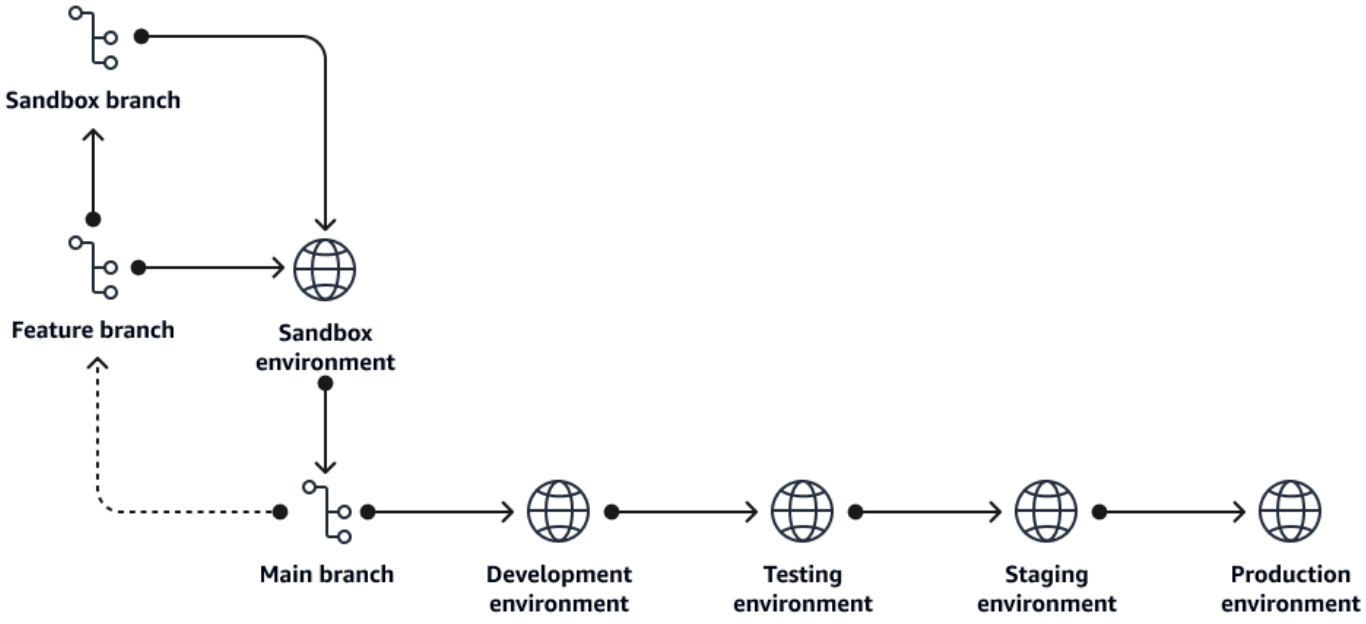
Diagram berikut dapat digunakan seperti kotak Punnett (Wikipedia) untuk memahami strategi percabangan Trunk. Sejajarkan cabang pada sumbu vertikal dengan AWS lingkungan pada sumbu horizontal untuk menentukan tindakan apa yang harus dilakukan dalam setiap skenario. Angka yang dilingkari memandu Anda melalui urutan tindakan yang diwakili dalam diagram. Diagram ini menunjukkan alur kerja pengembangan strategi percabangan Trunk, dari `feature` cabang di

lingkungan kotak pasir hingga pelepasan produksi cabang. `main` Untuk informasi lebih lanjut tentang aktivitas yang terjadi di setiap lingkungan, lihat DevOps lingkungan dalam panduan ini.




## Cabang dalam strategi Trunk

Strategi percabangan batang biasanya memiliki cabang-cabang berikut.




## cabang fitur

Anda mengembangkan fitur atau membuat hotfix di `feature` cabang. Untuk membuat `feature` cabang, Anda bercabang dari `main` cabang. Pengembang mengulangi, melakukan, dan menguji kode di `feature` cabang. Ketika fitur selesai, pengembang mempromosikan fitur tersebut. Hanya ada dua jalur ke depan dari `feature` cabang:

- Gabungkan ke dalam cabang `sandbox`
- Buat permintaan gabungan ke cabang `main`

| | |
|---|---|
| Konvensi penamaan: | `feature/<story number>_<developer initials>_<descriptor>` |
| Contoh konvensi penamaan: | `feature/123456_MS_Implement_Feature_A` |

## cabang kotak pasir

Cabang ini adalah cabang batang non-standar, tetapi berguna untuk pengembangan CI/CD pipa. `sandbox`Cabang ini terutama digunakan untuk tujuan berikut:

- Lakukan penyebaran penuh ke lingkungan kotak pasir dengan menggunakan saluran pipa CI/CD
- Kembangkan dan uji pipa sebelum mengirimkan permintaan gabungan untuk pengujian penuh di lingkungan yang lebih rendah, seperti pengembangan atau pengujian.

`Sandbox`cabang bersifat sementara dan dimaksudkan untuk berumur pendek. Mereka harus dihapus setelah pengujian spesifik selesai.

| | |
|---|---|
| Konvensi penamaan: | `sandbox/<story number>_<developer initials>_<descriptor>` |
| Contoh konvensi penamaan: | `sandbox/123456_MS_Test_Pipeline_Deploy` |

### cabang utama

`main`Cabang selalu mewakili kode yang berjalan dalam produksi. Kode bercabang dari`main`, dikembangkan, dan kemudian digabungkan kembali ke. `main` Penerapan dari `main` dapat menargetkan lingkungan apa pun. Untuk melindungi dari penghapusan, aktifkan perlindungan cabang untuk cabang. `main`

| | |
|---|---|
| Konvensi penamaan: | `main` |

### cabang hotfix

Tidak ada `hotfix` cabang khusus dalam alur kerja berbasis batang. Hotfix menggunakan `feature` cabang.

## Keuntungan dan kerugian dari strategi Trunk

Strategi percabangan Trunk sangat cocok untuk tim pengembangan yang lebih kecil, matang, yang memiliki keterampilan komunikasi yang kuat. Ini juga berfungsi dengan baik jika Anda memiliki rilis fitur yang terus menerus dan bergulir untuk aplikasi. Ini tidak cocok jika Anda memiliki tim pengembangan yang besar atau terfragmentasi atau jika Anda memiliki rilis fitur terjadwal yang luas. Konflik gabungan akan terjadi dalam model ini, jadi ketahuilah bahwa penyelesaian konflik penggabungan adalah keterampilan utama. Semua anggota tim harus dilatih sesuai dengan itu.

## Keuntungan

Trunk-based pengembangan menawarkan beberapa keuntungan yang dapat meningkatkan proses pengembangan, merampingkan kolaborasi, dan meningkatkan kualitas perangkat lunak secara keseluruhan. Berikut ini adalah beberapa manfaat utama:

- Loop umpan balik yang lebih cepat — Dengan pengembangan berbasis batang, pengembang sering mengintegrasikan perubahan kode mereka, seringkali beberapa kali sehari. Hal ini memungkinkan umpan balik yang lebih cepat mengenai potensi masalah dan membantu pengembang mengidentifikasi dan memperbaiki masalah lebih cepat daripada yang mereka lakukan dalam model pengembangan berbasis fitur.
- Mengurangi konflik penggabungan — Dalam pengembangan berbasis batang, risiko konflik penggabungan yang besar dan rumit diminimalkan karena perubahan terintegrasi terus menerus. Ini membantu mempertahankan basis kode yang lebih bersih dan mengurangi jumlah waktu yang dihabiskan untuk menyelesaikan konflik. Menyelesaikan konflik dapat memakan waktu dan rawan kesalahan dalam pengembangan berbasis fitur.
- Kolaborasi yang ditingkatkan — Trunk-based pengembangan mendorong pengembang untuk bekerja sama di cabang yang sama, mempromosikan komunikasi dan kolaborasi yang lebih baik dalam tim. Hal ini dapat menyebabkan pemecahan masalah yang lebih cepat dan dinamika tim yang lebih kohesif.
- Ulasan kode yang lebih mudah — Karena perubahan kode lebih kecil dan lebih sering dalam pengembangan berbasis batang, akan lebih mudah untuk melakukan tinjauan kode secara menyeluruh. Perubahan yang lebih kecil umumnya lebih mudah dipahami dan ditinjau, yang mengarah pada identifikasi potensi masalah dan perbaikan yang lebih efektif.
- Integrasi dan pengiriman berkelanjutan - Trunk-based pengembangan mendukung prinsip-prinsip integrasi berkelanjutan dan pengiriman berkelanjutan (CI/CD). Dengan menjaga basis kode dalam keadaan yang dapat dirilis dan sering mengintegrasikan perubahan, tim dapat lebih mudah mengadopsi CI/CD praktik, yang mengarah pada siklus penerapan yang lebih cepat dan kualitas perangkat lunak yang ditingkatkan.
- Kualitas kode yang ditingkatkan - Dengan integrasi, pengujian, dan tinjauan kode yang sering, pengembangan berbasis batang dapat berkontribusi pada kualitas kode keseluruhan yang lebih baik. Pengembang dapat menangkap dan memperbaiki masalah lebih cepat, mengurangi kemungkinan utang teknis terakumulasi dari waktu ke waktu.
- Strategi percabangan yang disederhanakan — Trunk-based pengembangan menyederhanakan strategi percabangan dengan mengurangi jumlah cabang berumur panjang. Ini dapat mempermudah pengelolaan dan pemeliharaan basis kode, terutama untuk proyek atau tim besar.

## Kekurangan

Trunk-based pengembangan memang memiliki beberapa kelemahan, yang dapat berdampak pada proses pengembangan dan dinamika tim. Berikut ini adalah beberapa kelemahan penting:

- Isolasi terbatas — Karena semua pengembang bekerja di cabang yang sama, perubahan mereka langsung terlihat oleh semua orang di tim. Hal ini dapat menyebabkan gangguan atau konflik, menyebabkan efek samping yang tidak diinginkan atau merusak bangunan. Sebaliknya, pengembangan berbasis fitur mengisolasi perubahan yang lebih baik sehingga pengembang dapat bekerja lebih mandiri.
- Peningkatan tekanan pada pengujian — Trunk-based pengembangan bergantung pada integrasi berkelanjutan dan pengujian otomatis untuk menangkap masalah dengan cepat. Namun, pendekatan ini dapat memberikan banyak tekanan pada infrastruktur pengujian dan membutuhkan rangkaian pengujian yang terawat dengan baik. Jika pengujian tidak komprehensif atau dapat diandalkan, itu dapat menyebabkan masalah yang tidak terdeteksi di cabang utama.
- Kurang kontrol atas rilis — Trunk-based pengembangan bertujuan untuk menjaga basis kode dalam keadaan yang dapat dirilis secara terus menerus. Meskipun ini bisa menguntungkan, mungkin tidak selalu cocok untuk proyek dengan jadwal rilis yang ketat atau yang memerlukan fitur khusus untuk dirilis bersama. Feature-basedpengembangan memberikan kontrol lebih besar atas kapan dan bagaimana fitur dirilis.
- Code churn — Dengan pengembang yang terus-menerus mengintegrasikan perubahan ke cabang utama, pengembangan berbasis batang dapat menyebabkan peningkatan churn kode. Hal ini dapat menyulitkan pengembang untuk melacak status basis kode saat ini dan dapat menyebabkan kebingungan ketika mencoba memahami efek dari perubahan terbaru.
- Membutuhkan budaya tim yang kuat - Trunk-based pengembangan menuntut disiplin, komunikasi, dan kolaborasi tingkat tinggi di antara anggota tim. Ini bisa menjadi tantangan untuk dipertahankan, terutama di tim yang lebih besar atau ketika bekerja dengan pengembang yang kurang berpengalaman dengan pendekatan ini.
- Tantangan skalabilitas — Seiring bertambahnya ukuran tim pengembangan, jumlah perubahan kode yang diintegrasikan ke dalam cabang utama dapat meningkat dengan cepat. Hal ini dapat menyebabkan kerusakan build dan kegagalan pengujian yang lebih sering, sehingga sulit untuk menjaga basis kode dalam status yang dapat dirilis.

# GitHub Strategi percabangan aliran

GitHub Flow adalah alur kerja ringan berbasis cabang yang dikembangkan oleh. GitHub GitHub Flow didasarkan pada gagasan cabang fitur berumur pendek yang digabungkan ke cabang utama saat fitur selesai dan siap digunakan. Prinsip-prinsip utama GitHub Flow adalah:

- Percabangan ringan — Pengembang dapat membuat cabang fitur untuk pekerjaan mereka hanya dengan beberapa klik, meningkatkan kemampuan untuk berkolaborasi dan bereksperimen tanpa memengaruhi cabang utama.
- Penerapan berkelanjutan - Perubahan diterapkan segera setelah digabungkan ke cabang utama, yang memungkinkan umpan balik dan iterasi yang cepat.
- Permintaan gabungan — Pengembang menggunakan permintaan gabungan untuk memulai proses diskusi dan peninjauan sebelum menggabungkan perubahan mereka ke cabang utama.

Untuk informasi selengkapnya tentang GitHub Flow, lihat sumber daya berikut:

- Menerapkan strategi percabangan GitHub Flow untuk DevOps lingkungan multi-akun (Panduan AWS Preskriptif)
- GitHub Alur Quickstart (GitHub dokumentasi)
- Mengapa GitHub Mengalir? (Situs web GitHub Flow)

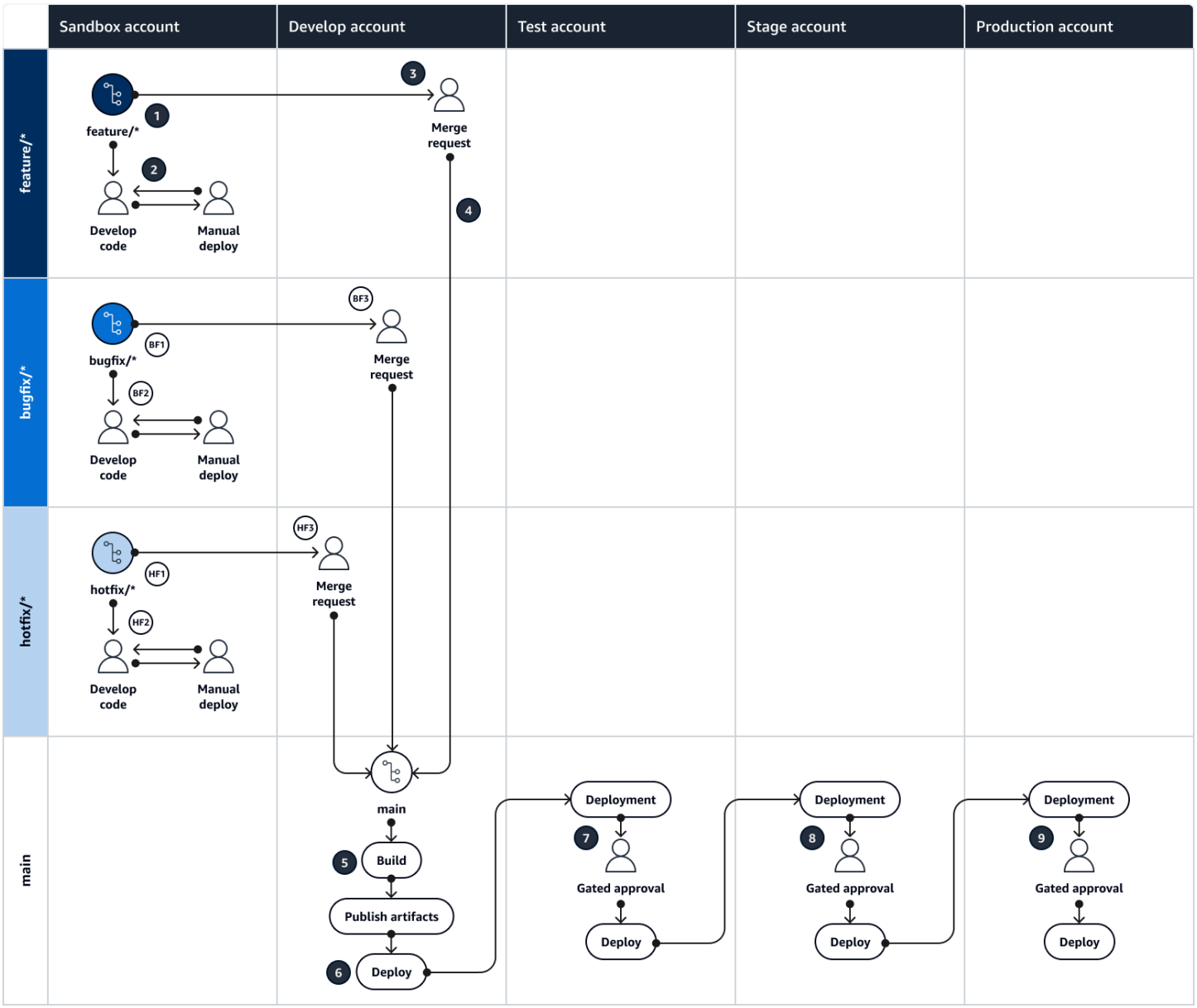
Topik di bagian ini:

- Gambaran visual dari strategi GitHub Flow
- Cabang dalam strategi GitHub Flow
- Keuntungan dan kerugian dari strategi GitHub Flow

## Gambaran visual dari strategi GitHub Flow

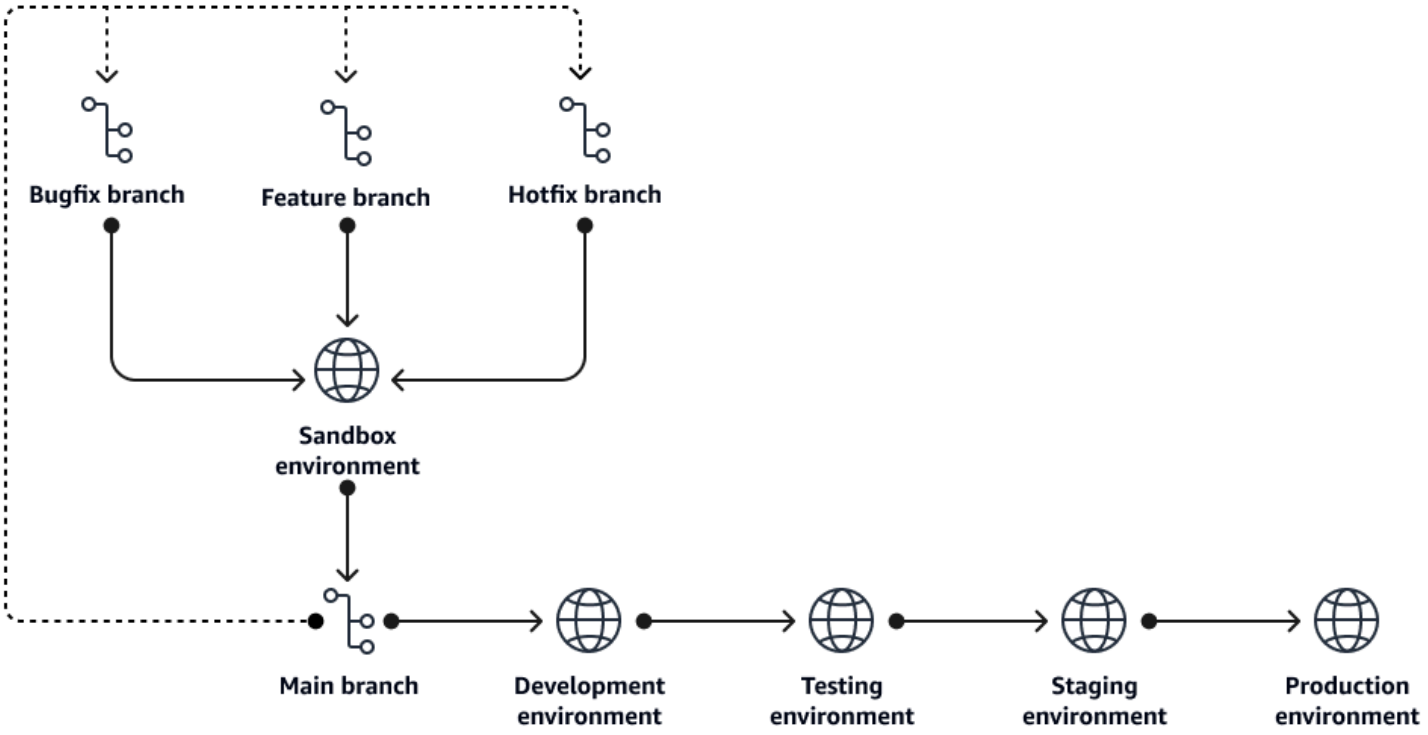
Diagram berikut dapat digunakan seperti kotak Punnett untuk memahami strategi percabangan GitHub Flow. Sejajarkan cabang pada sumbu vertikal dengan AWS lingkungan pada sumbu horizontal untuk menentukan tindakan apa yang harus dilakukan dalam setiap skenario. Angka yang dilingkari memandu Anda melalui urutan tindakan yang diwakili dalam diagram. Diagram ini menunjukkan alur kerja pengembangan strategi percabangan GitHub Flow, dari cabang fitur di

lingkungan kotak pasir hingga rilis produksi cabang utama. Untuk informasi lebih lanjut tentang aktivitas yang terjadi di setiap lingkungan, lihat DevOps lingkungan dalam panduan ini.




## Cabang dalam strategi GitHub Flow

Strategi percabangan GitHub Flow biasanya memiliki cabang-cabang berikut.




## cabang fitur

Anda mengembangkan fitur di `feature` cabang. Untuk membuat `feature` cabang, Anda bercabang dari `main` cabang. Pengembang mengulangi, melakukan, dan menguji kode di `feature` cabang. Saat fitur selesai, pengembang mempromosikan fitur tersebut dengan membuat permintaan gabungan ke`main`.

| | |
|---|---|
| Konvensi penamaan: | `feature/<story number>_<developer initials>_<descriptor>` |
| Contoh konvensi penamaan: | `feature/123456_MS_Implement_Feature_A` |

## cabang bugfix

`bugfix`Cabang digunakan untuk memperbaiki masalah. Cabang-cabang ini bercabang dari `main` cabang. Setelah perbaikan bug diuji di kotak pasir atau lingkungan yang lebih rendah, perbaikan ini dapat dipromosikan ke lingkungan yang lebih tinggi dengan menggabungkannya `main` melalui

permintaan gabungan. Ini adalah konvensi penamaan yang disarankan untuk organisasi dan pelacakan, proses ini juga dapat dikelola menggunakan cabang fitur.

| | |
|---|---|
| Konvensi penamaan: | `bugfix/<ticket number>_<developer initials>_<descriptor>` |
| Contoh konvensi penamaan: | `bugfix/123456_MS_Fix_Problem_A` |

### cabang hotfix

`hotfix`Cabang digunakan untuk menyelesaikan masalah kritis berdampak tinggi dengan penundaan minimal antara staf pengembangan dan kode yang digunakan dalam produksi. Cabang-cabang ini bercabang dari `main` cabang. Setelah perbaikan terbaru diuji di kotak pasir atau lingkungan yang lebih rendah, perbaikan terbaru dapat dipromosikan ke lingkungan yang lebih tinggi dengan menggabungkannya `main` melalui permintaan gabungan. Ini adalah konvensi penamaan yang disarankan untuk organisasi dan pelacakan, proses ini juga dapat dikelola menggunakan cabang fitur.

| | |
|---|---|
| Konvensi penamaan: | `hotfix/<ticket number>_<developer initials>_<descriptor>` |
| Contoh konvensi penamaan: | `hotfix/123456_MS_Fix_Problem_A` |

### cabang utama

`main`Cabang selalu mewakili kode yang berjalan dalam produksi. Kode digabungkan ke `main` cabang dari `feature` cabang dengan menggunakan permintaan gabungan. Untuk melindungi dari penghapusan dan mencegah pengembang mendorong kode secara langsung ke`main`, aktifkan perlindungan cabang untuk cabang. `main`

| | |
|---|---|
| Konvensi penamaan: | `main` |

# Keuntungan dan kerugian dari strategi GitHub Flow

Strategi percabangan Github Flow sangat cocok untuk tim pengembangan yang lebih kecil, matang, yang memiliki keterampilan komunikasi yang kuat. Strategi ini sangat cocok untuk tim yang ingin menerapkan pengiriman berkelanjutan, dan didukung dengan baik oleh CI/CD mesin umum. GitHub Flow ringan, tidak memiliki terlalu banyak aturan, dan mampu mendukung tim yang bergerak cepat. Ini tidak cocok jika tim Anda memiliki kepatuhan yang ketat atau proses rilis untuk diikuti. Konflik gabungan adalah hal biasa dalam model ini dan kemungkinan akan sering terjadi. Resolusi konflik gabungan adalah keterampilan utama, dan Anda harus melatih semua anggota tim yang sesuai.

## Keuntungan

GitHub Flow menawarkan beberapa keuntungan yang dapat meningkatkan proses pengembangan, merampingkan kolaborasi, dan meningkatkan kualitas perangkat lunak secara keseluruhan. Berikut ini adalah beberapa manfaat utama:

- Fleksibel dan ringan - GitHub Flow adalah alur kerja yang ringan dan fleksibel yang membantu pengembang berkolaborasi dalam proyek pengembangan perangkat lunak. Hal ini memungkinkan untuk iterasi cepat dan eksperimen dengan kompleksitas minimal.
- Kolaborasi yang disederhanakan - GitHub Flow menyediakan proses yang jelas dan efisien untuk mengelola pengembangan fitur. Ini mendorong perubahan kecil dan terfokus yang dapat dengan cepat ditinjau dan digabungkan, meningkatkan efisiensi.
- Kontrol versi yang jelas — Dengan GitHub Flow, setiap perubahan dilakukan di cabang terpisah. Ini menetapkan riwayat kontrol versi yang jelas dan dapat dilacak. Ini membantu pengembang melacak dan memahami perubahan, mengembalikan jika perlu, dan mempertahankan basis kode yang andal.
- Integrasi berkelanjutan yang mulus - GitHub Flow terintegrasi dengan alat integrasi berkelanjutan. Pembuatan permintaan tarik dapat memulai proses pengujian dan penerapan otomatis. Alat CI membantu Anda menguji perubahan secara menyeluruh sebelum digabungkan ke `main` cabang, mengurangi risiko memasukkan bug ke dalam basis kode.
- Umpan balik cepat dan peningkatan berkelanjutan - GitHub Flow mendorong loop umpan balik yang cepat dengan mempromosikan ulasan kode dan diskusi yang sering melalui permintaan tarik. Ini memfasilitasi deteksi dini masalah, mempromosikan berbagi pengetahuan di antara anggota tim, dan pada akhirnya mengarah pada kualitas kode yang lebih tinggi dan kolaborasi yang lebih baik dalam tim pengembangan.

- Rollback dan reverts yang disederhanakan - Jika perubahan kode menimbulkan bug atau masalah yang tidak terduga, GitHub Flow menyederhanakan proses memutar kembali atau mengembalikan perubahan. Dengan memiliki riwayat komitmen dan cabang yang jelas, lebih mudah untuk mengidentifikasi dan mengembalikan perubahan yang bermasalah, membantu mempertahankan basis kode yang stabil dan fungsional.
- Kurva belajar ringan — GitHub Flow dapat lebih mudah dipelajari dan diadopsi daripada Gitflow, terutama untuk tim yang sudah terbiasa dengan konsep Git dan kontrol versi. Kesederhanaan dan model percabangan yang intuitif membuatnya dapat diakses oleh pengembang dari berbagai tingkat pengalaman, mengurangi kurva pembelajaran yang terkait dengan mengadopsi alur kerja pengembangan baru.
- Pengembangan berkelanjutan — GitHub Flow memberdayakan tim untuk merangkul pendekatan penerapan berkelanjutan dengan memungkinkan penyebaran segera setiap perubahan segera setelah digabungkan ke dalam cabang. `main` Proses yang disederhanakan ini menghilangkan penundaan yang tidak perlu dan memastikan bahwa pembaruan dan peningkatan terbaru dengan cepat tersedia bagi pengguna. Ini menghasilkan siklus pengembangan yang lebih gesit dan responsif.

## Kekurangan

Sementara GitHub Flow menawarkan beberapa keuntungan, penting untuk mempertimbangkan potensi kerugiannya juga:

- Kesesuaian terbatas untuk proyek besar — GitHub Flow mungkin tidak cocok untuk proyek skala besar dengan basis kode yang kompleks dan beberapa cabang fitur jangka panjang. Dalam kasus seperti itu, alur kerja yang lebih terstruktur, seperti Gitflow, mungkin memberikan kontrol yang lebih baik atas pengembangan bersamaan dan manajemen rilis.
- Kurangnya struktur rilis formal — GitHub Flow tidak secara eksplisit mendefinisikan proses rilis atau mendukung fitur seperti pembuatan versi, perbaikan terbaru, atau cabang pemeliharaan. Ini bisa menjadi batasan untuk proyek yang memerlukan manajemen rilis yang ketat atau membutuhkan dukungan dan pemeliharaan jangka panjang.
- Dukungan terbatas untuk perencanaan rilis jangka panjang — GitHub Flow berfokus pada cabang fitur berumur pendek, yang mungkin tidak selaras dengan proyek yang memerlukan perencanaan rilis jangka panjang, seperti yang memiliki peta jalan yang ketat atau dependensi fitur yang ekstensif. Mengelola jadwal rilis yang kompleks dapat menjadi tantangan dalam batasan Flow. GitHub

- Potensi konflik penggabungan yang sering terjadi — Karena GitHub Flow mendorong seringnya percabangan dan penggabungan, ada kemungkinan menghadapi konflik penggabungan, terutama dalam proyek dengan banyak aktivitas pembangunan. Menyelesaikan konflik ini dapat memakan waktu dan mungkin memerlukan upaya tambahan dari tim pengembangan.
- Kurangnya fase alur kerja formal - GitHub Alur tidak mendefinisikan fase eksplisit untuk pengembangan, seperti tahap kandidat alfa, beta, atau rilis. Hal ini dapat membuat lebih sulit untuk mengkomunikasikan keadaan proyek saat ini atau tingkat stabilitas cabang atau rilis yang berbeda.
- Dampak perubahan yang melanggar — Karena GitHub Flow sering mendorong penggabungan perubahan ke dalam `main` cabang, ada risiko lebih tinggi untuk memperkenalkan perubahan yang melanggar yang memengaruhi stabilitas basis kode. Peninjauan kode dan praktik pengujian yang ketat sangat penting untuk mengurangi risiko ini secara efektif.

## Strategi percabangan Gitflow

Gitflow adalah model percabangan yang melibatkan penggunaan beberapa cabang untuk memindahkan kode dari pengembangan ke produksi. Gitflow berfungsi dengan baik untuk tim yang memiliki siklus rilis terjadwal dan kebutuhan untuk mendefinisikan kumpulan fitur sebagai rilis. Pengembangan diselesaikan di cabang fitur individu yang digabungkan, dengan persetujuan, menjadi cabang pengembangan, yang digunakan untuk integrasi. Fitur-fitur di cabang ini dianggap siap untuk produksi. Ketika semua fitur yang direncanakan telah terakumulasi di cabang pengembangan, cabang rilis dibuat untuk penerapan ke lingkungan atas. Pemisahan ini meningkatkan kontrol atas perubahan mana yang bergerak ke lingkungan bernama mana pada jadwal yang ditentukan. Jika perlu, Anda dapat mempercepat proses ini menjadi model penerapan yang lebih cepat.

Untuk informasi selengkapnya tentang strategi percabangan Gitflow, lihat sumber daya berikut:

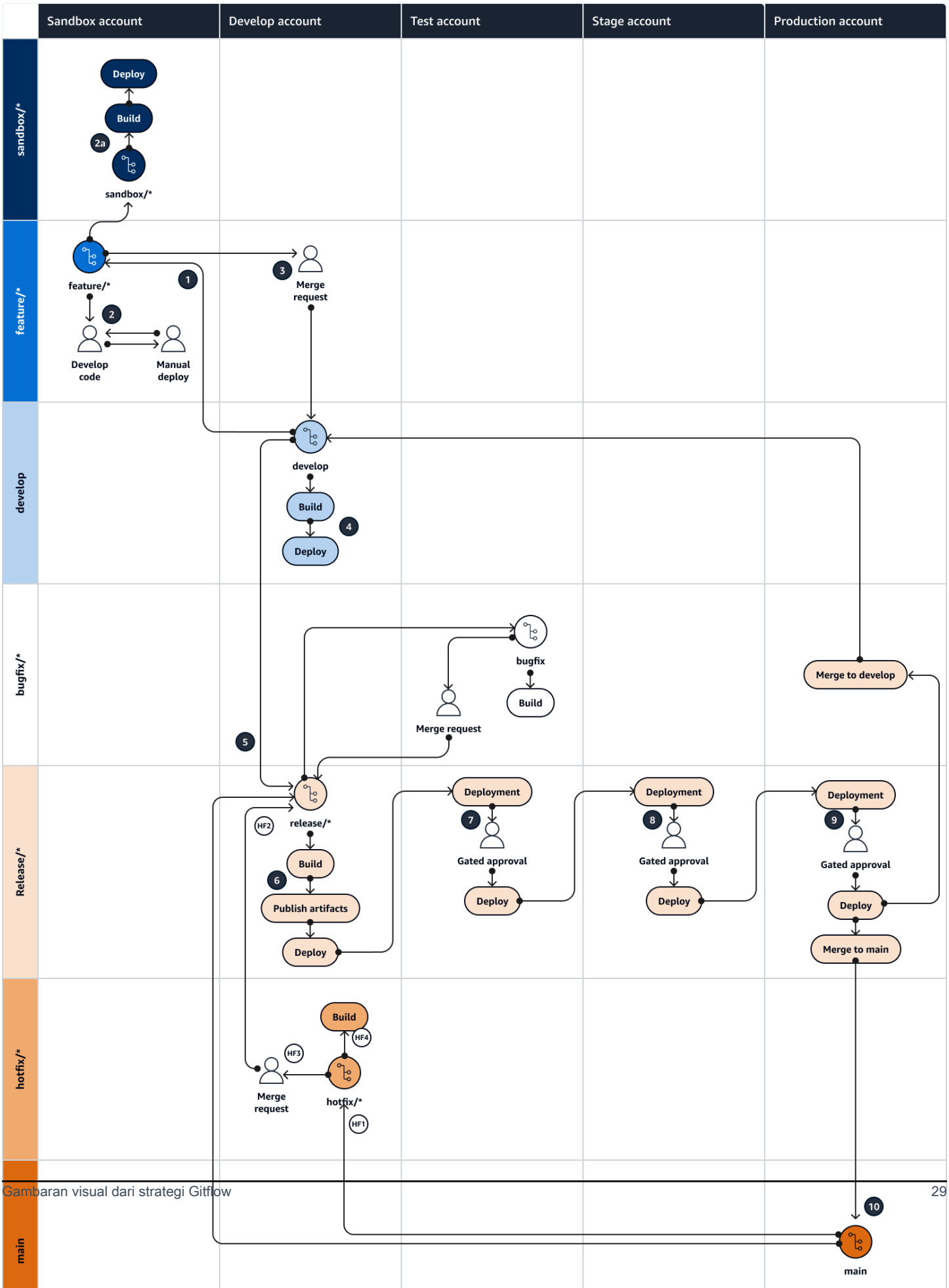
- Menerapkan strategi percabangan Gitflow untuk DevOps lingkungan multi-akun (AWS Panduan Preskriptif)
- Blog Gitflow asli (posting blog Vincent Driessen)
- Alur kerja Gitflow (Atlassian)

Topik di bagian ini:

- Gambaran visual dari strategi Gitflow
- Cabang dalam strategi Gitflow
- Keuntungan dan kerugian dari strategi Gitflow

## Gambaran visual dari strategi Gitflow

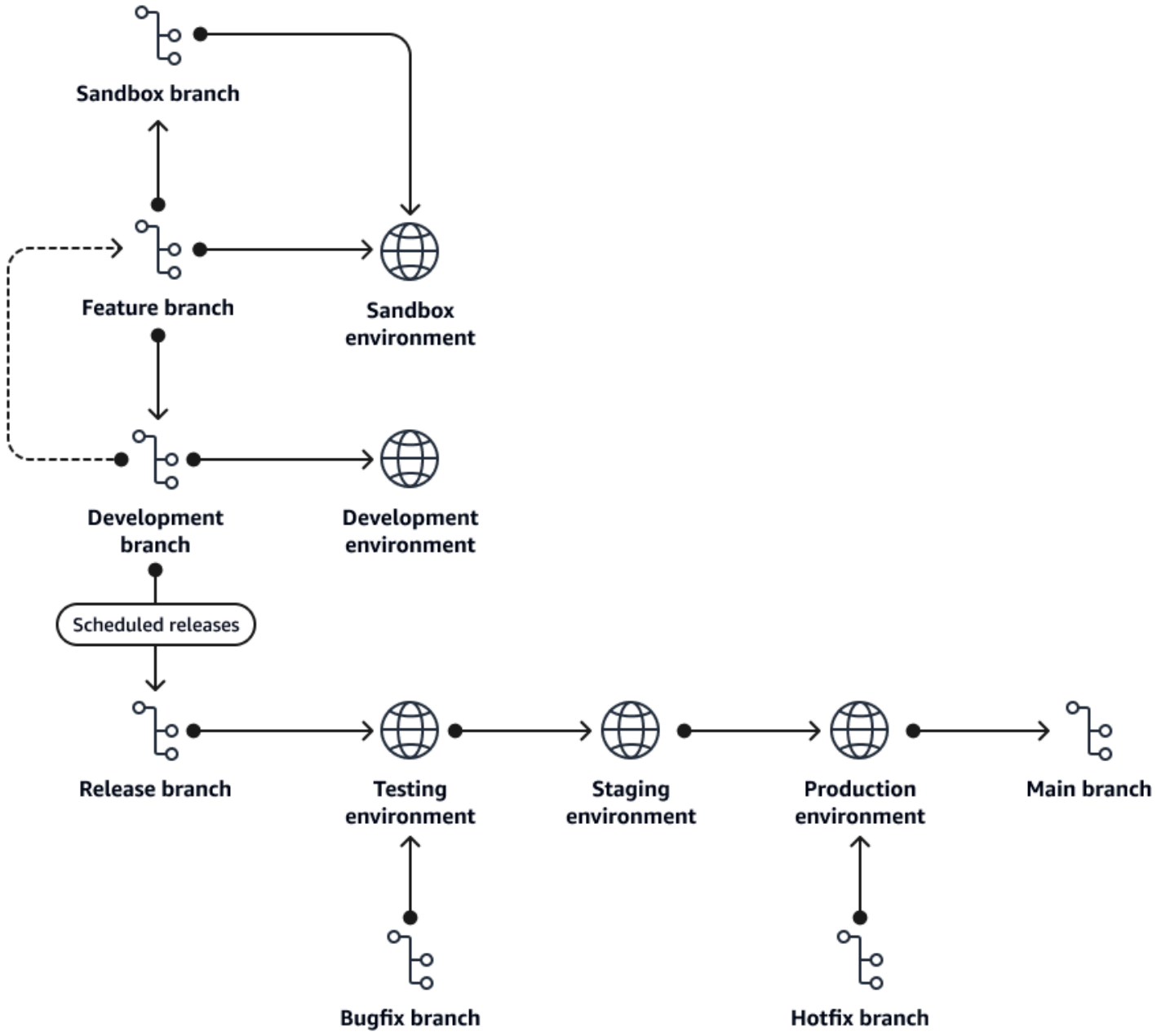
Diagram berikut dapat digunakan seperti kotak Punnett untuk memahami strategi percabangan Gitflow. Sejajarkan cabang pada sumbu vertikal dengan AWS lingkungan pada sumbu horizontal untuk menentukan tindakan apa yang harus dilakukan dalam setiap skenario. Angka yang dilingkari memandu Anda melalui urutan tindakan yang diwakili dalam diagram. Untuk informasi lebih lanjut tentang aktivitas yang terjadi di setiap lingkungan, lihat DevOps lingkungan dalam panduan ini.




Gambaran visual dari strategi Gitflow

# Cabang dalam strategi Gitflow

Strategi percabangan Gitflow biasanya memiliki cabang berikut.




## cabang fitur

`Feature`cabang adalah cabang jangka pendek tempat Anda mengembangkan fitur. `feature`Cabang dibuat dengan bercabang dari `develop` cabang. Pengembang mengulangi,

melakukan, dan menguji kode di `feature` cabang. Ketika fitur selesai, pengembang mempromosikan fitur tersebut. Hanya ada dua jalur ke depan dari cabang fitur:

- Gabungkan ke dalam cabang `sandbox`
- Buat permintaan gabungan ke cabang `develop`

| | |
|---|---|
| Konvensi penamaan: | `feature/<story number>_<developer initials>_<descriptor>` |
| Contoh konvensi penamaan: | `feature/123456_MS_Implement_Feature_A` |

## cabang kotak pasir

`sandbox`Cabang adalah cabang jangka pendek non-standar untuk Gitflow. Namun, ini berguna untuk pengembangan CI/CD pipa. `sandbox`Cabang ini terutama digunakan untuk tujuan berikut:

- Lakukan penyebaran penuh ke lingkungan kotak pasir dengan menggunakan CI/CD pipeline daripada penerapan manual.
- Kembangkan dan uji pipa sebelum mengirimkan permintaan gabungan untuk pengujian penuh di lingkungan yang lebih rendah, seperti pengembangan atau pengujian.

`Sandbox`cabang bersifat sementara dan tidak dimaksudkan untuk berumur panjang. Mereka harus dihapus setelah pengujian spesifik selesai.

| | |
|---|---|
| Konvensi penamaan: | `sandbox/<story number>_<developer initials>_<descriptor>` |
| Contoh konvensi penamaan: | `sandbox/123456_MS_Test_Pipeline_Deploy` |

## mengembangkan cabang

`develop`Cabang adalah cabang berumur panjang di mana fitur terintegrasi, dibangun, divalidasi, dan dikerahkan ke lingkungan pengembangan. Semua `feature` cabang digabung ke dalam `develop`

cabang. Penggabungan ke dalam `develop` cabang diselesaikan melalui permintaan gabungan yang memerlukan build yang berhasil dan dua persetujuan pengembang. Untuk mencegah penghapusan, aktifkan perlindungan cabang di cabang. `develop`

| | |
|---|---|
| Konvensi penamaan: | `develop` |

### cabang rilis

Di Gitflow, `release` cabang adalah cabang jangka pendek. Cabang-cabang ini istimewa karena Anda dapat menerapkannya ke beberapa lingkungan, merangkul metodologi build-once, deploy-many. `Release`cabang dapat menargetkan pengujian, pementasan, atau lingkungan produksi. Setelah tim pengembangan memutuskan untuk mempromosikan fitur ke lingkungan yang lebih tinggi, mereka membuat `release` cabang baru dan menggunakan penambahan nomor versi dari rilis sebelumnya. Di gerbang di setiap lingkungan, penerapan memerlukan persetujuan manual untuk melanjutkan. `Release`cabang harus membutuhkan permintaan penggabungan untuk diubah.

Setelah `release` cabang diterapkan ke produksi, cabang harus digabungkan kembali ke `main` cabang `develop` dan untuk memastikan bahwa perbaikan bug atau perbaikan terbaru digabungkan kembali ke upaya pengembangan masa depan.

| | |
|---|---|
| Konvensi penamaan: | `release/v{major}.{minor}` |
| Contoh konvensi penamaan: | `release/v1.0` |

### cabang utama

`main`Cabang adalah cabang berumur panjang yang selalu mewakili kode yang berjalan dalam produksi. Kode digabungkan ke `main` cabang secara otomatis dari cabang rilis setelah penerapan berhasil dari pipeline rilis. Untuk mencegah penghapusan, aktifkan perlindungan cabang di cabang. `main`

| | |
|---|---|
| Konvensi penamaan: | `main` |

### cabang bugfix

`bugfix`Cabang adalah cabang jangka pendek yang digunakan untuk memperbaiki masalah di cabang rilis yang belum dirilis ke produksi. `bugfix`Cabang hanya boleh digunakan untuk mempromosikan perbaikan di `release` cabang ke lingkungan pengujian, pementasan, atau produksi. `bugfix`Cabang selalu bercabang dari `release` cabang.

Setelah perbaikan bug diuji, itu dapat dipromosikan ke `release` cabang melalui permintaan gabungan. Kemudian Anda dapat mendorong `release` cabang ke depan dengan mengikuti proses rilis standar.

| | |
|---|---|
| Konvensi penamaan: | `bugfix/<ticket>_<developer initials>_<descriptor>` |
| Contoh konvensi penamaan: | `bugfix/123456_MS_Fix_Problem_A` |

### cabang hotfix

`hotfix`Cabang adalah cabang jangka pendek yang digunakan untuk memperbaiki masalah dalam produksi. Ini hanya digunakan untuk mempromosikan perbaikan yang harus dipercepat untuk mencapai lingkungan produksi. `hotfix`Cabang selalu bercabang dari`main`.

Setelah hotfix diuji, Anda dapat mempromosikannya ke produksi melalui permintaan gabungan ke `release` cabang yang dibuat dari. `main` Untuk pengujian, Anda kemudian dapat mendorong `release` cabang ke depan dengan mengikuti proses rilis standar.

| | |
|---|---|
| Konvensi penamaan: | `hotfix/<ticket>_<developer initials>_<descriptor>` |
| Contoh konvensi penamaan: | `hotfix/123456_MS_Fix_Problem_A` |

## Keuntungan dan kerugian dari strategi Gitflow

Strategi percabangan Gitflow sangat cocok untuk tim yang lebih besar dan lebih terdistribusi yang memiliki persyaratan rilis dan kepatuhan yang ketat. Gitflow berkontribusi pada siklus rilis yang dapat

diprediksi untuk organisasi, dan ini sering disukai oleh organisasi yang lebih besar. Gitflow juga cocok untuk tim yang membutuhkan pagar pembatas untuk menyelesaikan siklus pengembangan perangkat lunak mereka dengan benar. Ini karena ada banyak peluang untuk ulasan dan jaminan kualitas yang dibangun ke dalam strategi. Gitflow juga cocok untuk tim yang harus mempertahankan beberapa versi rilis produksi secara bersamaan. Beberapa kelemahan GitFlow adalah lebih kompleks daripada model percabangan lainnya dan membutuhkan kepatuhan yang ketat terhadap pola agar berhasil diselesaikan. Gitflow tidak bekerja dengan baik untuk organisasi yang berjuang untuk pengiriman berkelanjutan karena sifat kaku mengelola cabang rilis. Cabang rilis Gitflow dapat menjadi cabang berumur panjang yang dapat mengakumulasi utang teknis jika tidak ditangani dengan benar tepat waktu.

## Keuntungan

Gitflow-based pengembangan menawarkan beberapa keuntungan yang dapat meningkatkan proses pengembangan, merampingkan kolaborasi, dan meningkatkan kualitas perangkat lunak secara keseluruhan. Berikut ini adalah beberapa manfaat utama:

- Proses rilis yang dapat diprediksi — Gitflow mengikuti proses rilis reguler dan dapat diprediksi. Ini sangat cocok untuk tim dengan pengembangan reguler dan irama rilis.
- Kolaborasi yang ditingkatkan - Gitflow mendorong penggunaan `feature` dan `release` cabang. Kedua cabang ini membantu tim bekerja secara paralel dengan ketergantungan minimal satu sama lain.
- Sangat cocok untuk beberapa lingkungan — Gitflow menggunakan `release` cabang, yang bisa menjadi cabang yang berumur lebih panjang. Cabang-cabang ini memungkinkan tim untuk menargetkan rilis individu dalam jangka waktu yang lebih lama.
- Beberapa versi dalam produksi - Jika tim Anda mendukung beberapa versi perangkat lunak dalam produksi, `release` cabang Gitflow mendukung persyaratan ini.
- Built-in ulasan kualitas kode — Gitflow membutuhkan dan mendorong penggunaan ulasan kode dan persetujuan sebelum kode dipromosikan ke lingkungan lain. Proses ini menghilangkan gesekan antar pengembang dengan mengharuskan langkah ini untuk semua promosi kode.
- Manfaat organisasi — Gitflow memiliki keunggulan di tingkat organisasi juga. Gitflow mendorong penggunaan siklus rilis standar, yang membantu organisasi memahami dan mengantisipasi jadwal rilis. Karena bisnis sekarang memahami kapan fitur baru dapat dikirimkan, ada pengurangan gesekan tentang jadwal karena ada tanggal pengiriman yang ditetapkan.

## Kekurangan

Gitflow-based pengembangan memang memiliki beberapa kelemahan yang dapat berdampak pada proses pengembangan dan dinamika tim. Berikut ini adalah beberapa kelemahan penting:

- Kompleksitas — Gitflow adalah pola kompleks untuk dipelajari tim baru, dan Anda harus mematuhi aturan Gitflow agar berhasil menggunakannya.
- Penerapan berkelanjutan — Gitflow tidak sesuai dengan model di mana banyak penerapan dirilis ke produksi dengan cepat. Ini karena Gitflow memerlukan penggunaan beberapa cabang dan alur kerja yang ketat yang mengatur cabang. `release`
- Manajemen cabang — Gitflow menggunakan banyak cabang, yang dapat menjadi beban untuk dipertahankan. Mungkin sulit untuk melacak berbagai cabang dan menggabungkan kode yang dirilis untuk menjaga cabang tetap sejajar satu sama lain.
- Utang teknis — Karena rilis Gitflow biasanya lebih lambat daripada model percabangan lainnya, lebih banyak fitur dapat terakumulasi untuk rilis, yang dapat menyebabkan utang teknis menumpuk.

Tim harus hati-hati mempertimbangkan kelemahan ini ketika memutuskan apakah Gitflow-based pengembangan adalah pendekatan yang tepat untuk proyek mereka.

# Langkah selanjutnya

Panduan ini menjelaskan perbedaan antara tiga strategi percabangan Git yang umum: GitHub Flow, Gitflow, dan Trunk. Ini menjelaskan alur kerja mereka secara rinci dan juga memberikan kelebihan dan kekurangan masing-masing. Langkah selanjutnya adalah memilih salah satu alur kerja standar ini untuk organisasi Anda. Untuk menerapkan salah satu strategi percabangan ini, lihat yang berikut ini:

- Menerapkan strategi percabangan Trunk untuk lingkungan multi-akun DevOps
- Menerapkan strategi percabangan GitHub Flow untuk lingkungan multi-akun DevOps
- Menerapkan strategi percabangan Gitflow untuk lingkungan multi-akun DevOps

Jika Anda tidak yakin di mana harus memulai perjalanan tim Anda untuk menggunakan Git dan DevOps proses, kami sarankan memilih solusi standar dan mengujinya. Menggunakan konvensi percabangan standar membantu tim membangun dokumentasi yang ada dan mempelajari apa yang paling cocok untuk mereka.

Jangan takut untuk mengubah strategi Anda jika tidak bekerja untuk organisasi atau tim pengembangan Anda. Kebutuhan dan persyaratan tim pengembangan dapat berubah seiring waktu, dan tidak ada solusi tunggal yang sempurna.

# Sumber daya

Panduan ini tidak termasuk pelatihan untuk Git; namun, ada banyak sumber daya berkualitas tinggi yang tersedia di internet jika Anda memerlukan pelatihan ini. Kami menyarankan Anda memulai dengan situs dokumentasi Git.

Sumber daya berikut dapat membantu Anda dengan perjalanan percabangan Git Anda di. AWS Cloud

## AWS Panduan Preskriptif

- Menerapkan strategi percabangan Trunk untuk lingkungan multi-akun DevOps
- Menerapkan strategi percabangan GitHub Flow untuk lingkungan multi-akun DevOps
- Menerapkan strategi percabangan Gitflow untuk lingkungan multi-akun DevOps

## AWS Bimbingan lainnya

- AWS DevOps Bimbingan
- AWS Arsitektur Referensi Pipeline Deployment
- Apa itu DevOps?
- DevOpssumber daya

## Sumber daya lainnya

- Metodologi aplikasi dua belas faktor (12factor.net)
- Git-Rahasia () GitHub
- Gitflow
  - Blog Gitflow asli (posting blog Vincent Driessen)
  - Alur kerja Gitflow (Atlassian)
  - Gitflow on GitHub: Cara menggunakan alur kerja Git Flow dengan Repos GitHub Berbasis (video) YouTube
  - Contoh Init Aliran Git (YouTube video)

  - Cabang Rilis Gitflow dari Awal hingga Selesai (YouTube video)
- GitHub Aliran
  - GitHub Alur Quickstart (GitHub dokumentasi)
  - Mengapa GitHub Mengalir? (Situs web GitHub Flow)
- Batang
  - Pengantar Pengembangan Berbasis Trunk (situs web Pengembangan Berbasis Batang)

# Kontributor

## Mengotorisasi

- Mike Stephens, Arsitek Aplikasi Cloud Senior, AWS
- Rayjan Wilson, Arsitek Aplikasi Cloud Senior, AWS
- Abhilash Vinod, Ketua Tim, Arsitek Aplikasi Cloud Senior, AWS

## Meninjau

- Stephen DiCato, Konsultan Keamanan Senior, AWS
- Gaurav Samudra, Arsitek Aplikasi Cloud, AWS
- Steven Guggenheimer, Ketua Tim, Arsitek Aplikasi Cloud Senior, AWS

## Penulisan teknis

- Lilly AbouHarb, Penulis Teknis Senior, AWS

# Riwayat dokumen

Tabel berikut menjelaskan perubahan signifikan pada panduan ini. Jika Anda ingin diberi tahu tentang pembaruan masa depan, Anda dapat berlangganan umpan RSS.

| Perubahan | Deskripsi | Tanggal |
| --- | --- | --- |
| Perbarui | Kami mengganti gambar di Cabang dalam strategi Trunk, Cabang dalam strategi GitHub Flow, dan Cabang di bagian strategi Gitflow. | Juni 5, 2026 |
| Publikasi awal | — | Februari 15, 2024 |

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.