Memilih strategi percabangan Git untuk lingkungan multi-akun DevOps

AWS Bimbingan Preskriptif

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

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

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Merek dagang dan tampilan dagang Amazon tidak boleh digunakan sehubungan dengan produk atau layanan apa pun yang bukan milik Amazon, dengan cara apa pun yang dapat menyebabkan kebingungan di antara pelanggan, atau dengan cara apa pun yang merendahkan atau mendiskreditkan Amazon. Semua merek dagang lain yang tidak dimiliki oleh Amazon merupakan hak milik masing-masing pemiliknya, yang mungkin atau tidak terafiliasi, terkait dengan, atau disponsori oleh Amazon.

Table of Contents

Pengantar 1
Tujuan 1
Menggunakan praktik CI/CD 2
Memahami DevOps lingkungan 4
Lingkungan kotak pasir 5
Akses 5
Membangun langkah 5
Langkah-langkah penyebaran 6
Harapan sebelum pindah ke lingkungan pembangunan 6
Lingkungan pengembangan 6
Akses 5
Membangun langkah 5
Langkah-langkah penyebaran 6
Harapan sebelum pindah ke lingkungan pengujian 7
Lingkungan pengujian 8
Akses 5
Membangun langkah 5
Langkah-langkah penyebaran 6
Harapan sebelum pindah ke lingkungan pementasan 9
Lingkungan pementasan 9
Akses 5
Membangun langkah 5
Langkah-langkah penyebaran 6
Harapan sebelum pindah ke lingkungan produksi 10
Lingkungan produksi 10
Akses 5
Membangun langkah 5
Langkah-langkah penyebaran 6
Praktik terbaik untuk pengembangan berbasis Git 12
Strategi percabangan Git 14
Strategi percabangan batang 14
Gambaran visual dari strategi Trunk 15
Cabang dalam strategi Trunk 16
Keuntungan dan kerugian dari strategi Trunk 18

GitHub Strategi percabangan aliran 21
Gambaran visual dari strategi GitHub Flow 21
Cabang dalam strategi GitHub Flow 22
Keuntungan dan kerugian dari strategi GitHub Flow 24
Strategi percabangan Gitflow 27
Gambaran visual dari strategi Gitflow 28
Cabang dalam strategi Gitflow 30
Keuntungan dan kerugian dari strategi Gitflow 33
Langkah selanjutnya 36
Sumber daya 37
AWS Panduan Preskriptif 37
AWS Bimbingan lainnya 37
Sumber daya lainnya 37
Kontributor 39
Mengotorisasi 39
Meninjau 39
Penulisan teknis 39
Riwayat dokumen 40
Glosarium 41
..... 41
A 42
B 45
C 47
D 50
E 54
F 56
G 58
H 59
I 60
L 63
M 64
O 69
P 71
Q 74
R 75
D 78

T 82
U 83
V 84
W 84
Z 85
..... lxxxvii

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 praktik CI/CD

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 pengembangan
- Lingkungan pengujian
- Lingkungan pementasan
- Lingkungan produksi

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 day-to-day operasi. 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 percabangan berbasis GIT berikut secara rinci:

- Trunk — Pengembangan berbasis batang adalah praktik pengembangan perangkat lunak di mana semua pengembang bekerja pada satu cabang, biasanya disebut cabang atau. `trunk` `main` 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 cabang fitur individual. 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

Pengembangan berbasis Trunk adalah praktik pengembangan perangkat lunak di mana semua pengembang bekerja pada satu cabang, biasanya disebut cabang atau. `trunk` `main` 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.

Pengembangan berbasis batang 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 Pengembangan Berbasis Trunk (situs web Pengembangan Berbasis Batang)

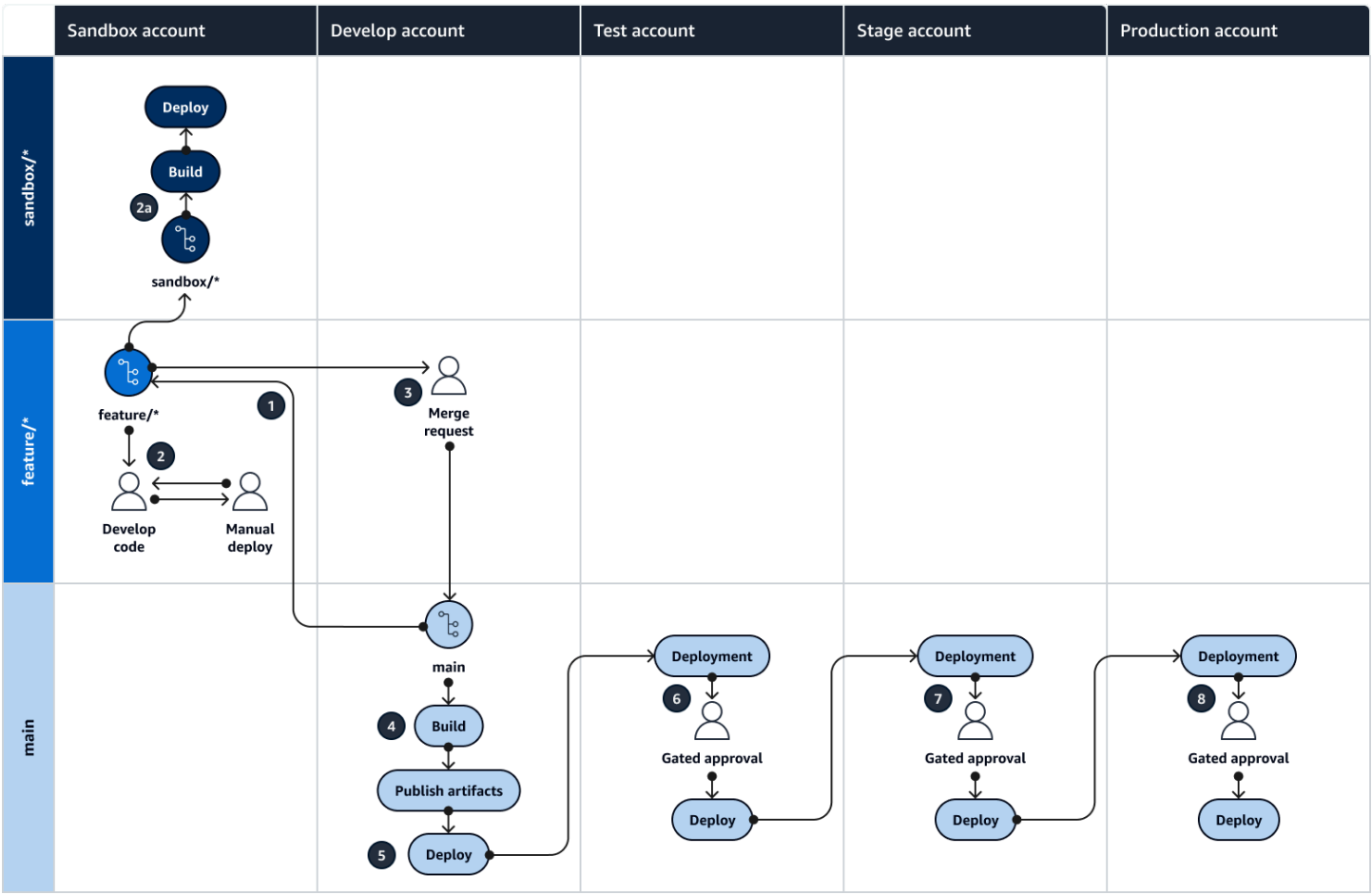
Topik di bagian ini:

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

Gambaran visual dari strategi Trunk

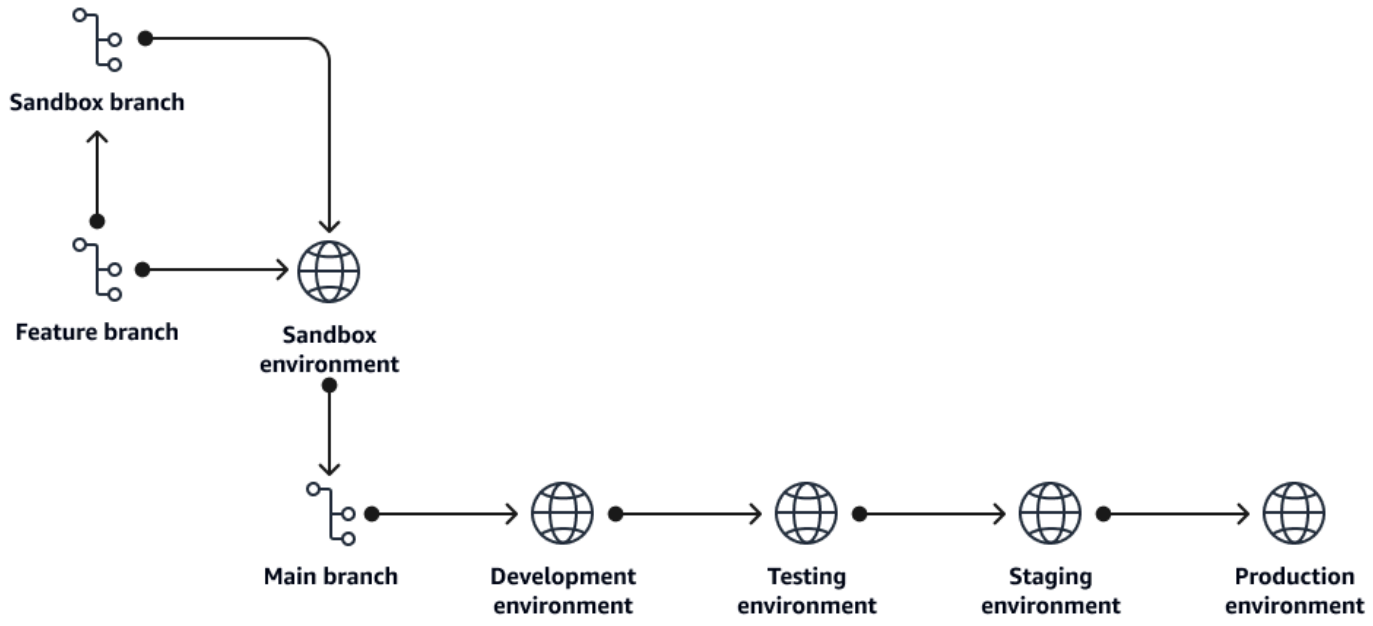
Diagram berikut dapat digunakan seperti kotak Punnett (Wikipedia) untuk memahami strategi percabangan Trunk. Sejajarkan cabang pada sumbu vertikal dengan AWS lingkungan pada sumbu horizontal untuk menentukan tindakan apa yang harus dilakukan dalam setiap skenario. Angka yang dilingkari memandu Anda melalui 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 trunk non-standar, tetapi berguna untuk pengembangan pipa CI/CD. `sandbox`Cabang ini terutama digunakan untuk tujuan berikut:

- Lakukan penyebaran penuh ke lingkungan kotak pasir dengan menggunakan 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

Pengembangan berbasis Trunk 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 — Pengembangan berbasis Trunk 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 - Pengembangan berbasis Trunk 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 praktik CI/CD, 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 — Pengembangan berbasis batang menyederhanakan strategi percabangan dengan mengurangi jumlah cabang yang berumur panjang. Ini dapat mempermudah pengelolaan dan pemeliharaan basis kode, terutama untuk proyek atau tim besar.

Kekurangan

Pengembangan berbasis Trunk 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 - Pengembangan berbasis Trunk 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 terpelihara dengan baik. Jika pengujian tidak komprehensif atau dapat diandalkan, itu dapat menyebabkan masalah yang tidak terdeteksi di cabang utama.
- Kurang kontrol atas rilis - Pengembangan berbasis Trunk 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. Pengembangan berbasis fitur 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 — Pengembangan berbasis Trunk 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 GitHubFlow didasarkan pada gagasan cabang fitur berumur pendek yang digabungkan ke dalam 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)

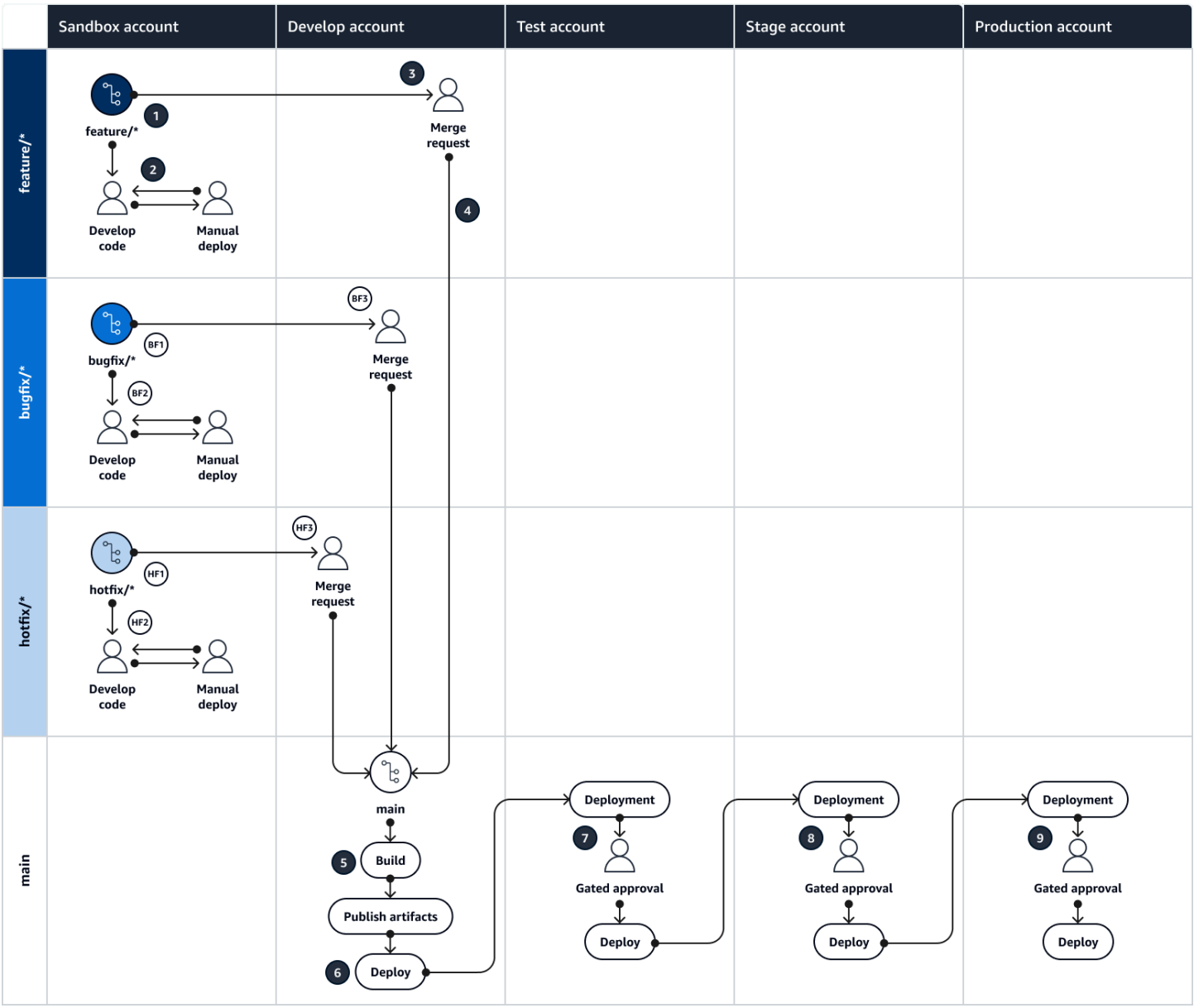
Topik di bagian ini:

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

Gambaran visual dari strategi GitHub Flow

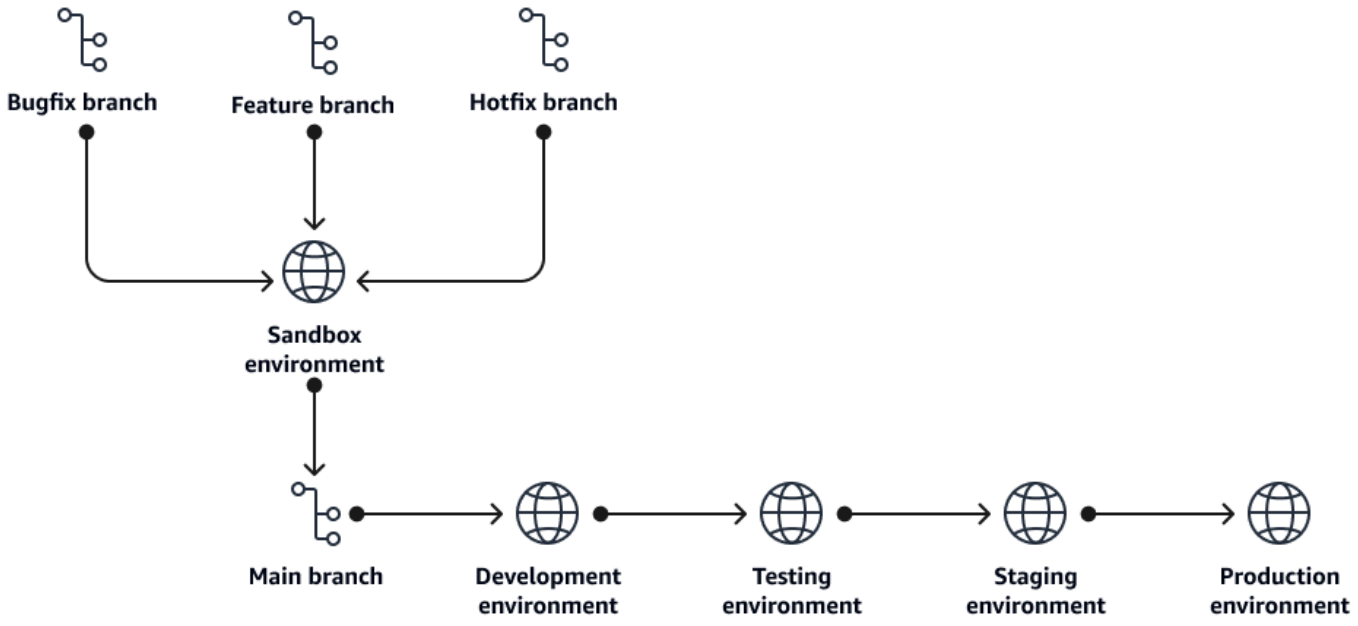
Diagram berikut dapat digunakan seperti kotak Punnett untuk memahami strategi percabangan GitHub Flow. Sejajarkan cabang pada sumbu vertikal dengan AWS lingkungan pada sumbu horizontal untuk menentukan tindakan apa yang harus dilakukan dalam setiap skenario. Angka yang dilingkari memandu Anda melalui 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 umumnya 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 mesin CI/CD 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. Praktik peninjauan kode dan 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:

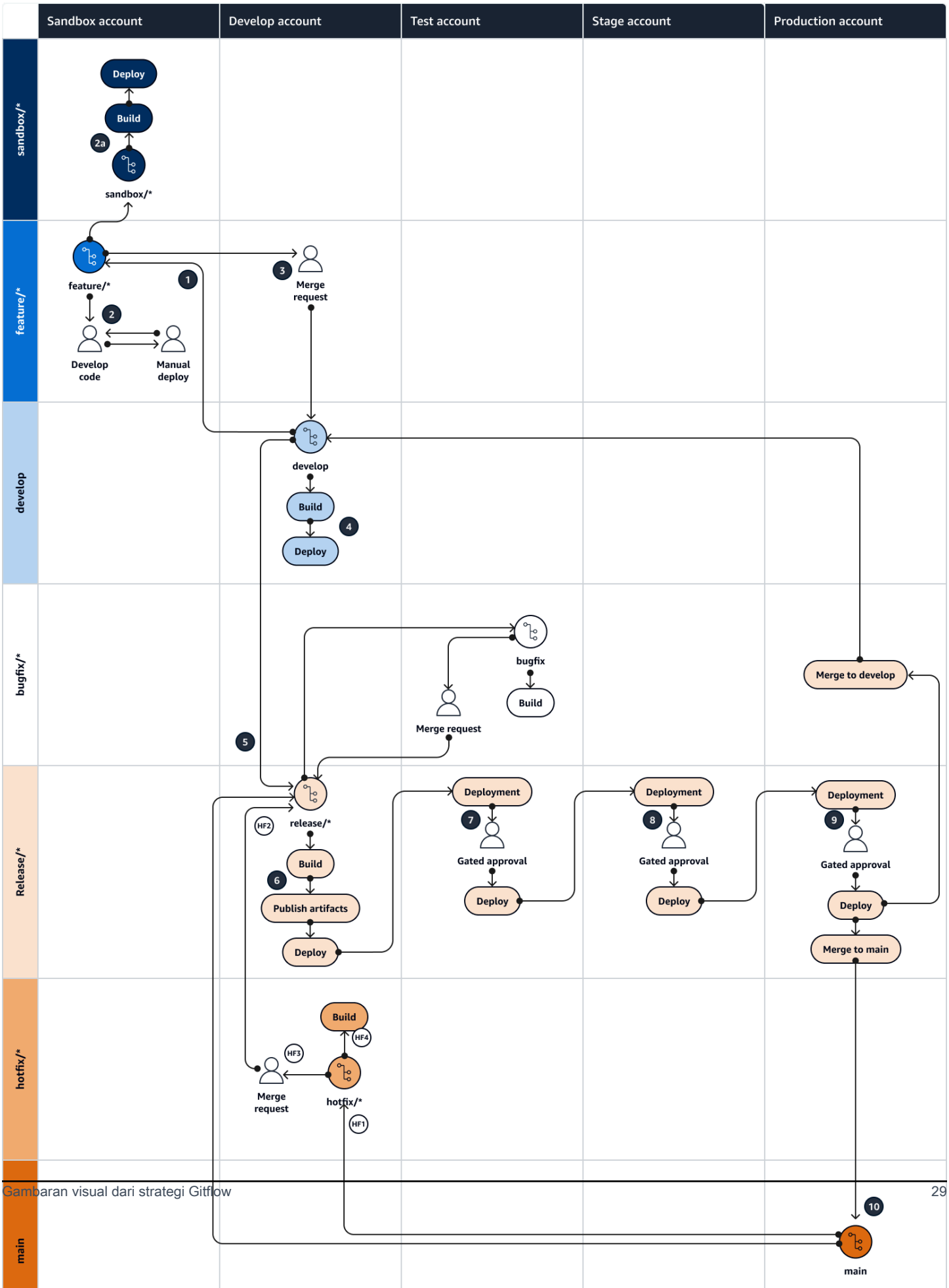
- Menerapkan strategi percabangan Gitflow untuk DevOps lingkungan multi-akun (AWS Panduan Preskriptif)
- Blog Gitflow asli (posting blog Vincent Driessen)
- Alur kerja Gitflow (Atlassian)

Topik di bagian ini:

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

Gambaran visual dari strategi Gitflow

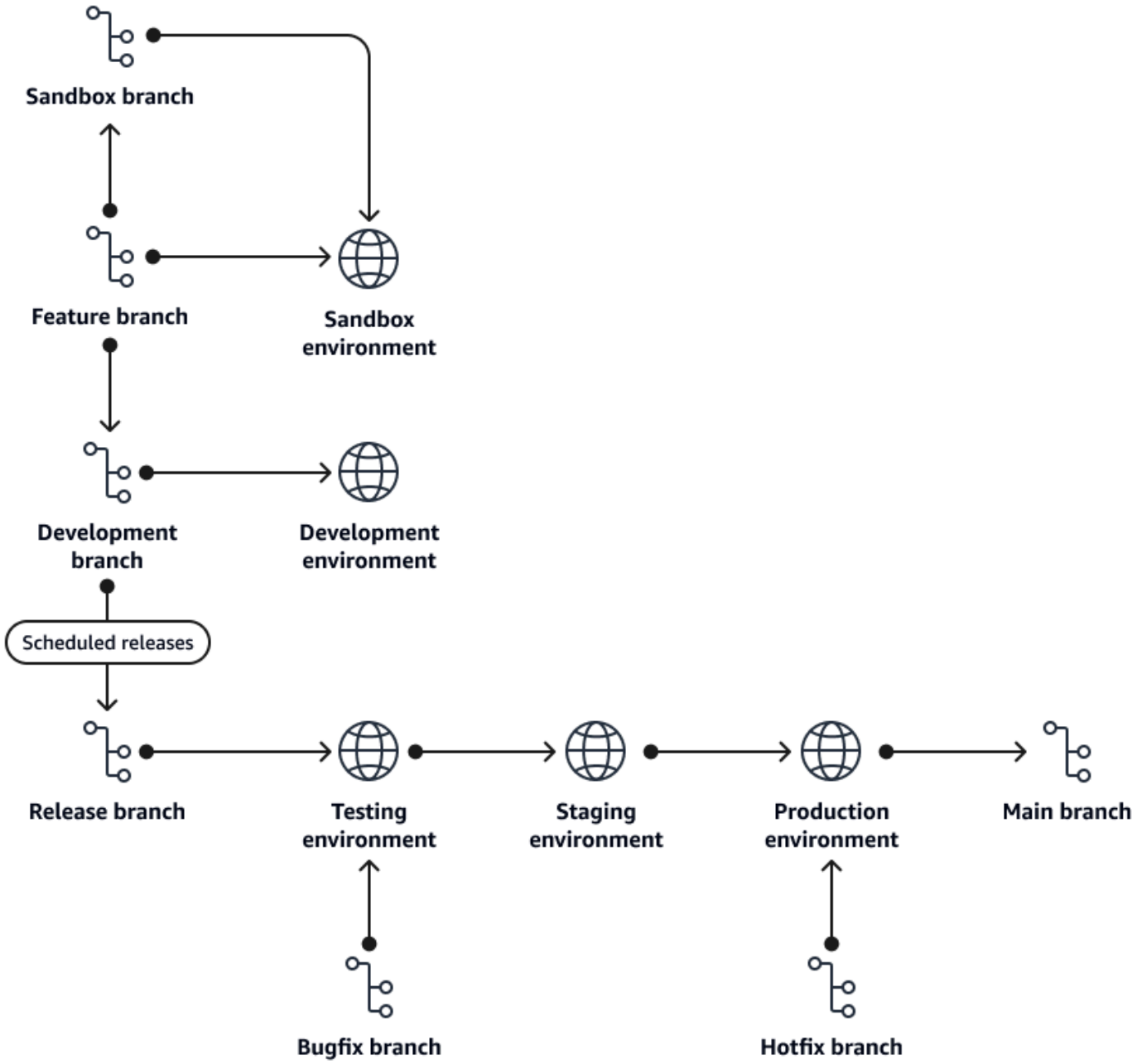
Diagram berikut dapat digunakan seperti kotak Punnett untuk memahami strategi percabangan Gitflow. Sejajarkan cabang pada sumbu vertikal dengan AWS lingkungan pada sumbu horizontal untuk menentukan tindakan apa yang harus dilakukan dalam setiap skenario. Angka yang dilingkari memandu Anda melalui urutan 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 pipa CI/CD. `sandbox`Cabang ini terutama digunakan untuk tujuan berikut:

- Lakukan penyebaran penuh ke lingkungan kotak pasir dengan menggunakan pipeline CI/CD 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

Pengembangan berbasis GitFlow 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.
- Ulasan kualitas kode bawaan — 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

Pengembangan berbasis Gitflow 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 mempertimbangkan dengan cermat kelemahan ini ketika memutuskan apakah pengembangan berbasis GitFlow 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
Publikasi awal	—	Februari 15, 2024

AWS Glosarium Panduan Preskriptif

Berikut ini adalah istilah yang umum digunakan dalam strategi, panduan, dan pola yang disediakan oleh Panduan AWS Preskriptif. Untuk menyarankan entri, silakan gunakan tautan Berikan umpan balik di akhir glosarium.

Nomor

7 Rs

Tujuh strategi migrasi umum untuk memindahkan aplikasi ke cloud. Strategi ini dibangun di atas 5 Rs yang diidentifikasi Gartner pada tahun 2011 dan terdiri dari yang berikut:

- Refactor/Re-Architect — Memindahkan aplikasi dan memodifikasi arsitekturnya dengan memanfaatkan sepenuhnya fitur cloud-native untuk meningkatkan kelincahan, kinerja, dan skalabilitas. Ini biasanya melibatkan porting sistem operasi dan database. Contoh: Migrasikan database Oracle lokal Anda ke Amazon Aurora PostgreSQL Compatible Edition.
- Replatform (angkat dan bentuk ulang) — Pindahkan aplikasi ke cloud, dan perkenalkan beberapa tingkat pengoptimalan untuk memanfaatkan kemampuan cloud. Contoh: Memigrasikan database Oracle lokal Anda ke Amazon Relational Database Service (Amazon RDS) untuk Oracle di. AWS Cloud
- Pembelian kembali (drop and shop) - Beralih ke produk yang berbeda, biasanya dengan beralih dari lisensi tradisional ke model SaaS. Contoh: Migrasikan sistem manajemen hubungan pelanggan (CRM) Anda ke Salesforce.com.
- Rehost (lift dan shift) — Pindahkan aplikasi ke cloud tanpa membuat perubahan apa pun untuk memanfaatkan kemampuan cloud. Contoh: Migrasikan database Oracle lokal Anda ke Oracle pada instance EC2 di. AWS Cloud
- Relokasi (hypervisor-level lift and shift) — Pindahkan infrastruktur ke cloud tanpa membeli perangkat keras baru, menulis ulang aplikasi, atau memodifikasi operasi yang ada. Anda memigrasikan server dari platform lokal ke layanan cloud untuk platform yang sama. Contoh: Migrasikan Microsoft Hyper-V aplikasi ke AWS.
- Pertahankan (kunjungi kembali) - Simpan aplikasi di lingkungan sumber Anda. Ini mungkin termasuk aplikasi yang memerlukan refactoring besar, dan Anda ingin menunda pekerjaan itu sampai nanti, dan aplikasi lama yang ingin Anda pertahankan, karena tidak ada pembenaran bisnis untuk memigrasikannya.

- Pensiun — Menonaktifkan atau menghapus aplikasi yang tidak lagi diperlukan di lingkungan sumber Anda.

A

ABAC

Lihat kontrol akses berbasis atribut.

layanan abstrak

Lihat layanan terkelola.

ASAM

Lihat atomisitas, konsistensi, isolasi, daya tahan.

migrasi aktif-aktif

Metode migrasi database di mana database sumber dan target tetap sinkron (dengan menggunakan alat replikasi dua arah atau operasi penulisan ganda), dan kedua database menangani transaksi dari menghubungkan aplikasi selama migrasi. Metode ini mendukung migrasi dalam batch kecil yang terkontrol alih-alih memerlukan pemotongan satu kali. Ini lebih fleksibel tetapi membutuhkan lebih banyak pekerjaan daripada migrasi aktif-pasif.

migrasi aktif-pasif

Metode migrasi database di mana database sumber dan target disimpan dalam sinkron, tetapi hanya database sumber yang menangani transaksi dari menghubungkan aplikasi sementara data direplikasi ke database target. Basis data target tidak menerima transaksi apa pun selama migrasi.

fungsi agregat

Fungsi SQL yang beroperasi pada sekelompok baris dan menghitung nilai pengembalian tunggal untuk grup. Contoh fungsi agregat meliputi `SUM` dan`MAX`.

AI

Lihat kecerdasan buatan.

AIOps

Lihat operasi kecerdasan buatan.

anonimisasi

Proses menghapus informasi pribadi secara permanen dalam kumpulan data. Anonimisasi dapat membantu melindungi privasi pribadi. Data anonim tidak lagi dianggap sebagai data pribadi.

anti-pola

Solusi yang sering digunakan untuk masalah berulang di mana solusinya kontra-produktif, tidak efektif, atau kurang efektif daripada alternatif.

kontrol aplikasi

Pendekatan keamanan yang memungkinkan penggunaan hanya aplikasi yang disetujui untuk membantu melindungi sistem dari malware.

portofolio aplikasi

Kumpulan informasi rinci tentang setiap aplikasi yang digunakan oleh organisasi, termasuk biaya untuk membangun dan memelihara aplikasi, dan nilai bisnisnya. Informasi ini adalah kunci untuk penemuan portofolio dan proses analisis dan membantu mengidentifikasi dan memprioritaskan aplikasi yang akan dimigrasi, dimodernisasi, dan dioptimalkan.

kecerdasan buatan (AI)

Bidang ilmu komputer yang didedikasikan untuk menggunakan teknologi komputasi untuk melakukan fungsi kognitif yang biasanya terkait dengan manusia, seperti belajar, memecahkan masalah, dan mengenali pola. Untuk informasi lebih lanjut, lihat Apa itu Kecerdasan Buatan?

operasi kecerdasan buatan (AIOps)

Proses menggunakan teknik pembelajaran mesin untuk memecahkan masalah operasional, mengurangi insiden operasional dan intervensi manusia, dan meningkatkan kualitas layanan. Untuk informasi selengkapnya tentang cara AIOps digunakan dalam strategi AWS migrasi, lihat panduan integrasi operasi.

enkripsi asimetris

Algoritma enkripsi yang menggunakan sepasang kunci, kunci publik untuk enkripsi dan kunci pribadi untuk dekripsi. Anda dapat berbagi kunci publik karena tidak digunakan untuk dekripsi, tetapi akses ke kunci pribadi harus sangat dibatasi.

atomisitas, konsistensi, isolasi, daya tahan (ACID)

Satu set properti perangkat lunak yang menjamin validitas data dan keandalan operasional database, bahkan dalam kasus kesalahan, kegagalan daya, atau masalah lainnya.

kontrol akses berbasis atribut (ABAC)

Praktik membuat izin berbutir halus berdasarkan atribut pengguna, seperti departemen, peran pekerjaan, dan nama tim. Untuk informasi selengkapnya, lihat ABAC untuk AWS dokumentasi AWS Identity and Access Management (IAM).

sumber data otoritatif

Lokasi di mana Anda menyimpan versi utama data, yang dianggap sebagai sumber informasi yang paling dapat diandalkan. Anda dapat menyalin data dari sumber data otoritatif ke lokasi lain untuk tujuan memproses atau memodifikasi data, seperti menganonimkan, menyunting, atau membuat nama samaran.

Zona Ketersediaan

Lokasi berbeda di dalam Wilayah AWS yang terisolasi dari kegagalan di Availability Zone lainnya dan menyediakan konektivitas jaringan latensi rendah yang murah ke Availability Zone lainnya di Wilayah yang sama.

AWS Kerangka Adopsi Cloud (AWS CAF)

Kerangka pedoman dan praktik terbaik AWS untuk membantu organisasi mengembangkan rencana yang efisien dan efektif untuk bergerak dengan sukses ke cloud. AWS CAF mengatur panduan ke dalam enam area fokus yang disebut perspektif: bisnis, orang, tata kelola, platform, keamanan, dan operasi. Perspektif bisnis, orang, dan tata kelola fokus pada keterampilan dan proses bisnis; perspektif platform, keamanan, dan operasi fokus pada keterampilan dan proses teknis. Misalnya, perspektif masyarakat menargetkan pemangku kepentingan yang menangani sumber daya manusia (SDM), fungsi kepegawaian, dan manajemen orang. Untuk perspektif ini, AWS CAF memberikan panduan untuk pengembangan, pelatihan, dan komunikasi orang untuk membantu mempersiapkan organisasi untuk adopsi cloud yang sukses. Untuk informasi lebih lanjut, lihat situs web AWS CAF dan whitepaper AWS CAF.

AWS Kerangka Kualifikasi Beban Kerja (AWS WQF)

Alat yang mengevaluasi beban kerja migrasi database, merekomendasikan strategi migrasi, dan memberikan perkiraan kerja. AWS WQF disertakan dengan AWS Schema Conversion Tool ()AWS SCT. Ini menganalisis skema database dan objek kode, kode aplikasi, dependensi, dan karakteristik kinerja, dan memberikan laporan penilaian.

B

bot buruk

Bot yang dimaksudkan untuk mengganggu atau membahayakan individu atau organisasi.

BCP

Lihat perencanaan kontinuitas bisnis.

grafik perilaku

Pandangan interaktif yang terpadu tentang perilaku dan interaksi sumber daya dari waktu ke waktu. Anda dapat menggunakan grafik perilaku dengan Amazon Detective untuk memeriksa upaya logon yang gagal, panggilan API yang mencurigakan, dan tindakan serupa. Untuk informasi selengkapnya, lihat Data dalam grafik perilaku di dokumentasi Detektif.

sistem big-endian

Sistem yang menyimpan byte paling signifikan terlebih dahulu. Lihat juga endianness.

klasifikasi biner

Sebuah proses yang memprediksi hasil biner (salah satu dari dua kelas yang mungkin). Misalnya, model ML Anda mungkin perlu memprediksi masalah seperti “Apakah email ini spam atau bukan spam?” atau “Apakah produk ini buku atau mobil?”

filter mekar

Struktur data probabilistik dan efisien memori yang digunakan untuk menguji apakah suatu elemen adalah anggota dari suatu himpunan.

deployment biru/hijau

Strategi penyebaran tempat Anda membuat dua lingkungan yang terpisah namun identik. Anda menjalankan versi aplikasi saat ini di satu lingkungan (biru) dan versi aplikasi baru di lingkungan lain (hijau). Strategi ini membantu Anda dengan cepat memutar kembali dengan dampak minimal.

bot

Aplikasi perangkat lunak yang menjalankan tugas otomatis melalui internet dan mensimulasikan aktivitas atau interaksi manusia. Beberapa bot berguna atau bermanfaat, seperti perayap web yang mengindeks informasi di internet. Beberapa bot lain, yang dikenal sebagai bot buruk, dimaksudkan untuk mengganggu atau membahayakan individu atau organisasi.

botnet

Jaringan bot yang terinfeksi oleh malware dan berada di bawah kendali satu pihak, yang dikenal sebagai bot herder atau operator bot. Botnet adalah mekanisme paling terkenal untuk skala bot dan dampaknya.

cabang

Area berisi repositori kode. Cabang pertama yang dibuat dalam repositori adalah cabang utama. Anda dapat membuat cabang baru dari cabang yang ada, dan Anda kemudian dapat mengembangkan fitur atau memperbaiki bug di cabang baru. Cabang yang Anda buat untuk membangun fitur biasanya disebut sebagai cabang fitur. Saat fitur siap dirilis, Anda menggabungkan cabang fitur kembali ke cabang utama. Untuk informasi selengkapnya, lihat Tentang cabang (GitHub dokumentasi).

akses break-glass

Dalam keadaan luar biasa dan melalui proses yang disetujui, cara cepat bagi pengguna untuk mendapatkan akses ke Akun AWS yang biasanya tidak memiliki izin untuk mengaksesnya. Untuk informasi lebih lanjut, lihat indikator Implementasikan prosedur break-glass dalam panduan Well-Architected AWS .

strategi brownfield

Infrastruktur yang ada di lingkungan Anda. Saat mengadopsi strategi brownfield untuk arsitektur sistem, Anda merancang arsitektur di sekitar kendala sistem dan infrastruktur saat ini. Jika Anda memperluas infrastruktur yang ada, Anda dapat memadukan strategi brownfield dan greenfield.

cache penyangga

Area memori tempat data yang paling sering diakses disimpan.

kemampuan bisnis

Apa yang dilakukan bisnis untuk menghasilkan nilai (misalnya, penjualan, layanan pelanggan, atau pemasaran). Arsitektur layanan mikro dan keputusan pengembangan dapat didorong oleh kemampuan bisnis. Untuk informasi selengkapnya, lihat bagian Terorganisir di sekitar kemampuan bisnis dari Menjalankan layanan mikro kontainer di whitepaper. AWS

perencanaan kelangsungan bisnis (BCP)

Rencana yang membahas dampak potensial dari peristiwa yang mengganggu, seperti migrasi skala besar, pada operasi dan memungkinkan bisnis untuk melanjutkan operasi dengan cepat.

C

KAFE

Lihat Kerangka Adopsi AWS Cloud.

penyebaran kenari

Rilis versi yang lambat dan bertahap untuk pengguna akhir. Ketika Anda yakin, Anda menyebarkan versi baru dan mengganti versi saat ini secara keseluruhan.

CCoE

Lihat Cloud Center of Excellence.

CDC

Lihat mengubah pengambilan data.

ubah pengambilan data (CDC)

Proses melacak perubahan ke sumber data, seperti tabel database, dan merekam metadata tentang perubahan tersebut. Anda dapat menggunakan CDC untuk berbagai tujuan, seperti mengaudit atau mereplikasi perubahan dalam sistem target untuk mempertahankan sinkronisasi.

rekayasa kekacauan

Dengan sengaja memperkenalkan kegagalan atau peristiwa yang mengganggu untuk menguji ketahanan sistem. Anda dapat menggunakan AWS Fault Injection Service (AWS FIS) untuk melakukan eksperimen yang menekankan AWS beban kerja Anda dan mengevaluasi responsnya.

CI/CD

Lihat integrasi berkelanjutan dan pengiriman berkelanjutan.

klasifikasi

Proses kategorisasi yang membantu menghasilkan prediksi. Model ML untuk masalah klasifikasi memprediksi nilai diskrit. Nilai diskrit selalu berbeda satu sama lain. Misalnya, model mungkin perlu mengevaluasi apakah ada mobil dalam gambar atau tidak.

Enkripsi sisi klien

Enkripsi data secara lokal, sebelum target Layanan AWS menerimanya.

Pusat Keunggulan Cloud (CCoE)

Tim multi-disiplin yang mendorong upaya adopsi cloud di seluruh organisasi, termasuk mengembangkan praktik terbaik cloud, memobilisasi sumber daya, menetapkan jadwal migrasi, dan memimpin organisasi melalui transformasi skala besar. Untuk informasi selengkapnya, lihat posting CCo E di Blog Strategi AWS Cloud Perusahaan.

komputasi cloud

Teknologi cloud yang biasanya digunakan untuk penyimpanan data jarak jauh dan manajemen perangkat IoT. Cloud computing umumnya terhubung ke teknologi edge computing.

model operasi cloud

Dalam organisasi TI, model operasi yang digunakan untuk membangun, mematangkan, dan mengoptimalkan satu atau lebih lingkungan cloud. Untuk informasi selengkapnya, lihat Membangun Model Operasi Cloud Anda.

tahap adopsi cloud

Empat fase yang biasanya dilalui organisasi ketika mereka bermigrasi ke AWS Cloud:

- Proyek — Menjalankan beberapa proyek terkait cloud untuk bukti konsep dan tujuan pembelajaran
- Foundation — Melakukan investasi dasar untuk meningkatkan adopsi cloud Anda (misalnya, membuat landing zone, mendefinisikan CCo E, membuat model operasi)
- Migrasi — Migrasi aplikasi individual
- Re-invention — Mengoptimalkan produk dan layanan, dan berinovasi di cloud

Tahapan ini didefinisikan oleh Stephen Orban dalam posting blog The Journey Toward Cloud-First & the Stages of Adoption di blog Strategi Perusahaan. AWS Cloud Untuk informasi tentang bagaimana kaitannya dengan strategi AWS migrasi, lihat panduan kesiapan migrasi.

CMDB

Lihat database manajemen konfigurasi.

repositori kode

Lokasi di mana kode sumber dan aset lainnya, seperti dokumentasi, sampel, dan skrip, disimpan dan diperbarui melalui proses kontrol versi. Repositori cloud umum termasuk GitHub atau. Bitbucket Cloud Setiap versi kode disebut cabang. Dalam struktur layanan mikro, setiap repositori

dikhususkan untuk satu bagian fungsionalitas. Pipa CI/CD tunggal dapat menggunakan beberapa repositori.

cache dingin

Cache buffer yang kosong, tidak terisi dengan baik, atau berisi data basi atau tidak relevan. Ini mempengaruhi kinerja karena instance database harus membaca dari memori utama atau disk, yang lebih lambat daripada membaca dari cache buffer.

data dingin

Data yang jarang diakses dan biasanya historis. Saat menanyakan jenis data ini, kueri lambat biasanya dapat diterima. Memindahkan data ini ke tingkat atau kelas penyimpanan yang berkinerja lebih rendah dan lebih murah dapat mengurangi biaya.

visi komputer (CV)

Bidang AI yang menggunakan pembelajaran mesin untuk menganalisis dan mengekstrak informasi dari format visual seperti gambar dan video digital. Misalnya, Amazon SageMaker AI menyediakan algoritma pemrosesan gambar untuk CV.

konfigurasi drift

Untuk beban kerja, konfigurasi berubah dari status yang diharapkan. Ini dapat menyebabkan beban kerja menjadi tidak patuh, dan biasanya bertahap dan tidak disengaja.

database manajemen konfigurasi (CMDB)

Repositori yang menyimpan dan mengelola informasi tentang database dan lingkungan TI, termasuk komponen perangkat keras dan perangkat lunak dan konfigurasinya. Anda biasanya menggunakan data dari CMDB dalam penemuan portofolio dan tahap analisis migrasi.

paket kesesuaian

Kumpulan AWS Config aturan dan tindakan remediasi yang dapat Anda kumpulkan untuk menyesuaikan kepatuhan dan pemeriksaan keamanan Anda. Anda dapat menerapkan paket kesesuaian sebagai entitas tunggal di Akun AWS dan Region, atau di seluruh organisasi, dengan menggunakan templat YAMM. Untuk informasi selengkapnya, lihat Paket kesesuaian dalam dokumentasi. AWS Config

integrasi berkelanjutan dan pengiriman berkelanjutan (CI/CD)

Proses mengotomatiskan sumber, membangun, menguji, pementasan, dan tahap produksi dari proses rilis perangkat lunak. CI/CD is commonly described as a pipeline. CI/CDdapat membantu

Anda mengotomatiskan proses, meningkatkan produktivitas, meningkatkan kualitas kode, dan memberikan lebih cepat. Untuk informasi lebih lanjut, lihat Manfaat pengiriman berkelanjutan. CD juga dapat berarti penerapan berkelanjutan. Untuk informasi selengkapnya, lihat Continuous Delivery vs Continuous Deployment.

CV

Lihat visi komputer.

D

data saat istirahat

Data yang stasioner di jaringan Anda, seperti data yang ada di penyimpanan.

klasifikasi data

Proses untuk mengidentifikasi dan mengkategorikan data dalam jaringan Anda berdasarkan kekritisan dan sensitivitasnya. Ini adalah komponen penting dari setiap strategi manajemen risiko keamanan siber karena membantu Anda menentukan perlindungan dan kontrol retensi yang tepat untuk data. Klasifikasi data adalah komponen pilar keamanan dalam AWS Well-Architected Framework. Untuk informasi selengkapnya, lihat Klasifikasi data.

penyimpangan data

Variasi yang berarti antara data produksi dan data yang digunakan untuk melatih model ML, atau perubahan yang berarti dalam data input dari waktu ke waktu. Penyimpangan data dapat mengurangi kualitas, akurasi, dan keadilan keseluruhan dalam prediksi model ML.

data dalam transit

Data yang aktif bergerak melalui jaringan Anda, seperti antara sumber daya jaringan.

jala data

Kerangka arsitektur yang menyediakan kepemilikan data terdistribusi dan terdesentralisasi dengan manajemen dan tata kelola terpusat.

minimalisasi data

Prinsip pengumpulan dan pemrosesan hanya data yang sangat diperlukan. Mempraktikkan minimalisasi data di dalamnya AWS Cloud dapat mengurangi risiko privasi, biaya, dan jejak karbon analitik Anda.

perimeter data

Satu set pagar pembatas pencegahan di AWS lingkungan Anda yang membantu memastikan bahwa hanya identitas tepercaya yang mengakses sumber daya tepercaya dari jaringan yang diharapkan. Untuk informasi selengkapnya, lihat Membangun perimeter data pada AWS.

prapemrosesan data

Untuk mengubah data mentah menjadi format yang mudah diuraikan oleh model ML Anda. Preprocessing data dapat berarti menghapus kolom atau baris tertentu dan menangani nilai yang hilang, tidak konsisten, atau duplikat.

asal data

Proses melacak asal dan riwayat data sepanjang siklus hidupnya, seperti bagaimana data dihasilkan, ditransmisikan, dan disimpan.

subjek data

Individu yang datanya dikumpulkan dan diproses.

gudang data

Sistem manajemen data yang mendukung intelijen bisnis, seperti analitik. Gudang data biasanya berisi sejumlah besar data historis, dan biasanya digunakan untuk kueri dan analisis.

bahasa definisi database (DDL)

Pernyataan atau perintah untuk membuat atau memodifikasi struktur tabel dan objek dalam database.

bahasa manipulasi basis data (DHTML)

Pernyataan atau perintah untuk memodifikasi (memasukkan, memperbarui, dan menghapus) informasi dalam database.

DDL

Lihat bahasa definisi database.

ansambel yang dalam

Untuk menggabungkan beberapa model pembelajaran mendalam untuk prediksi. Anda dapat menggunakan ansambel dalam untuk mendapatkan prediksi yang lebih akurat atau untuk memperkirakan ketidakpastian dalam prediksi.

pembelajaran mendalam

Subbidang ML yang menggunakan beberapa lapisan jaringan saraf tiruan untuk mengidentifikasi pemetaan antara data input dan variabel target yang diinginkan.

defense-in-depth

Pendekatan keamanan informasi di mana serangkaian mekanisme dan kontrol keamanan dilapisi dengan cermat di seluruh jaringan komputer untuk melindungi kerahasiaan, integritas, dan ketersediaan jaringan dan data di dalamnya. Saat Anda mengadopsi strategi ini AWS, Anda menambahkan beberapa kontrol pada lapisan AWS Organizations struktur yang berbeda untuk membantu mengamankan sumber daya. Misalnya, defense-in-depth pendekatan mungkin menggabungkan otentikasi multi-faktor, segmentasi jaringan, dan enkripsi.

administrator yang didelegasikan

Di AWS Organizations, layanan yang kompatibel dapat mendaftarkan akun AWS anggota untuk mengelola akun organisasi dan mengelola izin untuk layanan tersebut. Akun ini disebut administrator yang didelegasikan untuk layanan itu. Untuk informasi selengkapnya dan daftar layanan yang kompatibel, lihat Layanan yang berfungsi dengan AWS Organizations AWS Organizations dokumentasi.

deployment

Proses pembuatan aplikasi, fitur baru, atau perbaikan kode tersedia di lingkungan target. Deployment melibatkan penerapan perubahan dalam basis kode dan kemudian membangun dan menjalankan basis kode itu di lingkungan aplikasi.

lingkungan pengembangan

Lihat lingkungan.

kontrol detektif

Kontrol keamanan yang dirancang untuk mendeteksi, mencatat, dan memperingatkan setelah suatu peristiwa terjadi. Kontrol ini adalah garis pertahanan kedua, memperingatkan Anda tentang peristiwa keamanan yang melewati kontrol pencegahan yang ada. Untuk informasi selengkapnya, lihat Kontrol Detektif dalam Menerapkan kontrol keamanan pada. AWS

pemetaan aliran nilai pengembangan (DVSM)

Sebuah proses yang digunakan untuk mengidentifikasi dan memprioritaskan kendala yang mempengaruhi kecepatan dan kualitas dalam siklus hidup pengembangan perangkat lunak. DVSM memperluas proses pemetaan aliran nilai yang awalnya dirancang untuk praktik

manufaktur ramping. Ini berfokus pada langkah-langkah dan tim yang diperlukan untuk menciptakan dan memindahkan nilai melalui proses pengembangan perangkat lunak.

kembar digital

Representasi virtual dari sistem dunia nyata, seperti bangunan, pabrik, peralatan industri, atau jalur produksi. Kembar digital mendukung pemeliharaan prediktif, pemantauan jarak jauh, dan optimalisasi produksi.

tabel dimensi

Dalam skema bintang, tabel yang lebih kecil yang berisi atribut data tentang data kuantitatif dalam tabel fakta. Atribut tabel dimensi biasanya bidang teks atau angka diskrit yang berperilaku seperti teks. Atribut ini biasanya digunakan untuk pembatasan kueri, pemfilteran, dan pelabelan set hasil.

musibah

Peristiwa yang mencegah beban kerja atau sistem memenuhi tujuan bisnisnya di lokasi utama yang digunakan. Peristiwa ini dapat berupa bencana alam, kegagalan teknis, atau akibat dari tindakan manusia, seperti kesalahan konfigurasi yang tidak disengaja atau serangan malware.

pemulihan bencana (DR)

Strategi dan proses yang Anda gunakan untuk meminimalkan downtime dan kehilangan data yang disebabkan oleh bencana. Untuk informasi selengkapnya, lihat Disaster Recovery of Workloads on AWS: Recovery in the Cloud in the AWS Well-Architected Framework.

DML~

Lihat bahasa manipulasi basis data.

desain berbasis domain

Pendekatan untuk mengembangkan sistem perangkat lunak yang kompleks dengan menghubungkan komponennya ke domain yang berkembang, atau tujuan bisnis inti, yang dilayani oleh setiap komponen. Konsep ini diperkenalkan oleh Eric Evans dalam bukunya, Domain-Driven Design: Tackling Complexity in the Heart of Software (Boston: Addison-Wesley Professional, 2003). Untuk informasi tentang cara menggunakan desain berbasis domain dengan pola gambar pencekik, lihat Memodernisasi layanan web Microsoft ASP.NET (ASMX) lama secara bertahap menggunakan container dan Amazon API Gateway.

DR

Lihat pemulihan bencana.

deteksi drift

Melacak penyimpangan dari konfigurasi dasar. Misalnya, Anda dapat menggunakan AWS CloudFormation untuk mendeteksi penyimpangan dalam sumber daya sistem, atau Anda dapat menggunakannya AWS Control Tower untuk mendeteksi perubahan di landing zone yang mungkin memengaruhi kepatuhan terhadap persyaratan tata kelola.

DVSM

Lihat pemetaan aliran nilai pengembangan.

E

EDA

Lihat analisis data eksplorasi.

EDI

Lihat pertukaran data elektronik.

komputasi tepi

Teknologi yang meningkatkan daya komputasi untuk perangkat pintar di tepi jaringan IoT. Jika dibandingkan dengan komputasi awan, komputasi tepi dapat mengurangi latensi komunikasi dan meningkatkan waktu respons.

pertukaran data elektronik (EDI)

Pertukaran otomatis dokumen bisnis antar organisasi. Untuk informasi selengkapnya, lihat Apa itu Pertukaran Data Elektronik.

enkripsi

Proses komputasi yang mengubah data plaintext, yang dapat dibaca manusia, menjadi ciphertext.

kunci enkripsi

String kriptografi dari bit acak yang dihasilkan oleh algoritma enkripsi. Panjang kunci dapat bervariasi, dan setiap kunci dirancang agar tidak dapat diprediksi dan unik.

endianness

Urutan byte disimpan dalam memori komputer. Sistem big-endian menyimpan byte paling signifikan terlebih dahulu. Sistem little-endian menyimpan byte paling tidak signifikan terlebih dahulu.

titik akhir

Lihat titik akhir layanan.

layanan endpoint

Layanan yang dapat Anda host di cloud pribadi virtual (VPC) untuk dibagikan dengan pengguna lain. Anda dapat membuat layanan endpoint dengan AWS PrivateLink dan memberikan izin kepada prinsipal lain Akun AWS atau ke AWS Identity and Access Management (IAM). Akun atau prinsipal ini dapat terhubung ke layanan endpoint Anda secara pribadi dengan membuat titik akhir VPC antarmuka. Untuk informasi selengkapnya, lihat Membuat layanan titik akhir di dokumentasi Amazon Virtual Private Cloud (Amazon VPC).

perencanaan sumber daya perusahaan (ERP)

Sistem yang mengotomatiskan dan mengelola proses bisnis utama (seperti akuntansi, MES, dan manajemen proyek) untuk suatu perusahaan.

enkripsi amplop

Proses mengenkripsi kunci enkripsi dengan kunci enkripsi lain. Untuk informasi selengkapnya, lihat Enkripsi amplop dalam dokumentasi AWS Key Management Service (AWS KMS).

lingkungan

Sebuah contoh dari aplikasi yang sedang berjalan. Berikut ini adalah jenis lingkungan yang umum dalam komputasi awan:

- Development Environment — Sebuah contoh dari aplikasi yang berjalan yang hanya tersedia untuk tim inti yang bertanggung jawab untuk memelihara aplikasi. Lingkungan pengembangan digunakan untuk menguji perubahan sebelum mempromosikannya ke lingkungan atas. Jenis lingkungan ini kadang-kadang disebut sebagai lingkungan pengujian.
- lingkungan yang lebih rendah — Semua lingkungan pengembangan untuk aplikasi, seperti yang digunakan untuk build awal dan pengujian.
- lingkungan produksi — Sebuah contoh dari aplikasi yang berjalan yang pengguna akhir dapat mengakses. Dalam pipa CI/CD, lingkungan produksi adalah lingkungan penyebaran terakhir.

- lingkungan atas — Semua lingkungan yang dapat diakses oleh pengguna selain tim pengembangan inti. Ini dapat mencakup lingkungan produksi, lingkungan praproduksi, dan lingkungan untuk pengujian penerimaan pengguna.

epik

Dalam metodologi tangkas, kategori fungsional yang membantu mengatur dan memprioritaskan pekerjaan Anda. Epik memberikan deskripsi tingkat tinggi tentang persyaratan dan tugas implementasi. Misalnya, epos keamanan AWS CAF mencakup manajemen identitas dan akses, kontrol detektif, keamanan infrastruktur, perlindungan data, dan respons insiden. Untuk informasi selengkapnya tentang epos dalam strategi AWS migrasi, lihat panduan implementasi program.

ERP

Lihat perencanaan sumber daya perusahaan.

analisis data eksplorasi (EDA)

Proses menganalisis dataset untuk memahami karakteristik utamanya. Anda mengumpulkan atau mengumpulkan data dan kemudian melakukan penyelidikan awal untuk menemukan pola, mendeteksi anomali, dan memeriksa asumsi. EDA dilakukan dengan menghitung statistik ringkasan dan membuat visualisasi data.

F

tabel fakta

Tabel tengah dalam skema bintang. Ini menyimpan data kuantitatif tentang operasi bisnis. Biasanya, tabel fakta berisi dua jenis kolom: kolom yang berisi ukuran dan yang berisi kunci asing ke tabel dimensi.

gagal cepat

Filosofi yang menggunakan pengujian yang sering dan bertahap untuk mengurangi siklus hidup pengembangan. Ini adalah bagian penting dari pendekatan tangkas.

batas isolasi kesalahan

Dalam AWS Cloud, batas seperti Availability Zone, Wilayah AWS, control plane, atau data plane yang membatasi efek kegagalan dan membantu meningkatkan ketahanan beban kerja. Untuk informasi selengkapnya, lihat Batas Isolasi AWS Kesalahan.

cabang fitur

Lihat cabang.

fitur

Data input yang Anda gunakan untuk membuat prediksi. Misalnya, dalam konteks manufaktur, fitur bisa berupa gambar yang diambil secara berkala dari lini manufaktur.

pentingnya fitur

Seberapa signifikan fitur untuk prediksi model. Ini biasanya dinyatakan sebagai skor numerik yang dapat dihitung melalui berbagai teknik, seperti Shapley Additive Explanations (SHAP) dan gradien terintegrasi. Untuk informasi lebih lanjut, lihat Interpretabilitas model pembelajaran mesin dengan. AWS

transformasi fitur

Untuk mengoptimalkan data untuk proses ML, termasuk memperkaya data dengan sumber tambahan, menskalakan nilai, atau mengekstrak beberapa set informasi dari satu bidang data. Hal ini memungkinkan model ML untuk mendapatkan keuntungan dari data. Misalnya, jika Anda memecah tanggal “2021-05-27 00:15:37" menjadi “2021", “Mei”, “Kamis”, dan “15", Anda dapat membantu algoritme pembelajaran mempelajari pola bernuansa yang terkait dengan komponen data yang berbeda.

beberapa tembakan mendorong

Menyediakan LLM dengan sejumlah kecil contoh yang menunjukkan tugas dan output yang diinginkan sebelum memintanya untuk melakukan tugas serupa. Teknik ini adalah aplikasi pembelajaran dalam konteks, di mana model belajar dari contoh (bidikan) yang tertanam dalam petunjuk. Beberapa bidikan dapat efektif untuk tugas-tugas yang memerlukan pemformatan, penalaran, atau pengetahuan domain tertentu. Lihat juga bidikan nol.

FGAC

Lihat kontrol akses berbutir halus.

kontrol akses berbutir halus (FGAC)

Penggunaan beberapa kondisi untuk mengizinkan atau menolak permintaan akses.

migrasi flash-cut

Metode migrasi database yang menggunakan replikasi data berkelanjutan melalui pengambilan data perubahan untuk memigrasikan data dalam waktu sesingkat mungkin, alih-alih

menggunakan pendekatan bertahap. Tujuannya adalah untuk menjaga downtime seminimal mungkin.

FM

Lihat model pondasi.

model pondasi (FM)

Jaringan saraf pembelajaran mendalam yang besar yang telah melatih kumpulan data besar-besaran data umum dan tidak berlabel. FMs mampu melakukan berbagai tugas umum, seperti memahami bahasa, menghasilkan teks dan gambar, dan berbicara dalam bahasa alami. Untuk informasi selengkapnya, lihat Apa itu Model Foundation.

G

AI generatif

Subset model AI yang telah dilatih pada sejumlah besar data dan yang dapat menggunakan prompt teks sederhana untuk membuat konten dan artefak baru, seperti gambar, video, teks, dan audio. Untuk informasi lebih lanjut, lihat Apa itu AI Generatif.

pemblokiran geografis

Lihat pembatasan geografis.

pembatasan geografis (pemblokiran geografis)

Di Amazon CloudFront, opsi untuk mencegah pengguna di negara tertentu mengakses distribusi konten. Anda dapat menggunakan daftar izinkan atau daftar blokir untuk menentukan negara yang disetujui dan dilarang. Untuk informasi selengkapnya, lihat Membatasi distribusi geografis konten Anda dalam dokumentasi. CloudFront

Alur kerja Gitflow

Pendekatan di mana lingkungan bawah dan atas menggunakan cabang yang berbeda dalam repositori kode sumber. Alur kerja Gitflow dianggap warisan, dan alur kerja berbasis batang adalah pendekatan modern yang lebih disukai.

gambar emas

Sebuah snapshot dari sistem atau perangkat lunak yang digunakan sebagai template untuk menyebarkan instance baru dari sistem atau perangkat lunak itu. Misalnya, di bidang manufaktur,

gambar emas dapat digunakan untuk menyediakan perangkat lunak pada beberapa perangkat dan membantu meningkatkan kecepatan, skalabilitas, dan produktivitas dalam operasi manufaktur perangkat.

strategi greenfield

Tidak adanya infrastruktur yang ada di lingkungan baru. Saat mengadopsi strategi greenfield untuk arsitektur sistem, Anda dapat memilih semua teknologi baru tanpa batasan kompatibilitas dengan infrastruktur yang ada, juga dikenal sebagai brownfield. Jika Anda memperluas infrastruktur yang ada, Anda dapat memadukan strategi brownfield dan greenfield.

pagar pembatas

Aturan tingkat tinggi yang membantu mengatur sumber daya, kebijakan, dan kepatuhan di seluruh unit organisasi ()OUs. Pagar pembatas preventif menegakkan kebijakan untuk memastikan keselarasan dengan standar kepatuhan. Mereka diimplementasikan dengan menggunakan kebijakan kontrol layanan dan batas izin IAM. Detective guardrails mendeteksi pelanggaran kebijakan dan masalah kepatuhan, dan menghasilkan peringatan untuk remediasi. Mereka diimplementasikan dengan menggunakan AWS Config, AWS Security Hub, Amazon GuardDuty AWS Trusted Advisor, Amazon Inspector, dan pemeriksaan khusus AWS Lambda .

H

HA

Lihat ketersediaan tinggi.

migrasi database heterogen

Memigrasi database sumber Anda ke database target yang menggunakan mesin database yang berbeda (misalnya, Oracle ke Amazon Aurora). Migrasi heterogen biasanya merupakan bagian dari upaya arsitektur ulang, dan mengubah skema dapat menjadi tugas yang kompleks. AWS menyediakan AWS SCT yang membantu dengan konversi skema.

ketersediaan tinggi (HA)

Kemampuan beban kerja untuk beroperasi terus menerus, tanpa intervensi, jika terjadi tantangan atau bencana. Sistem HA dirancang untuk gagal secara otomatis, secara konsisten memberikan kinerja berkualitas tinggi, dan menangani beban dan kegagalan yang berbeda dengan dampak kinerja minimal.

modernisasi sejarawan

Pendekatan yang digunakan untuk memodernisasi dan meningkatkan sistem teknologi operasional (OT) untuk melayani kebutuhan industri manufaktur dengan lebih baik. Sejarawan adalah jenis database yang digunakan untuk mengumpulkan dan menyimpan data dari berbagai sumber di pabrik.

data penahanan

Sebagian dari data historis berlabel yang ditahan dari kumpulan data yang digunakan untuk melatih model pembelajaran mesin. Anda dapat menggunakan data penahanan untuk mengevaluasi kinerja model dengan membandingkan prediksi model dengan data penahanan.

migrasi database homogen

Memigrasi database sumber Anda ke database target yang berbagi mesin database yang sama (misalnya, Microsoft SQL Server ke Amazon RDS for SQL Server). Migrasi homogen biasanya merupakan bagian dari upaya rehosting atau replatforming. Anda dapat menggunakan utilitas database asli untuk memigrasi skema.

data panas

Data yang sering diakses, seperti data real-time atau data translasi terbaru. Data ini biasanya memerlukan tingkat atau kelas penyimpanan berkinerja tinggi untuk memberikan respons kueri yang cepat.

perbaikan terbaru

Perbaikan mendesak untuk masalah kritis dalam lingkungan produksi. Karena urgensinya, perbaikan terbaru biasanya dibuat di luar alur kerja DevOps rilis biasa.

periode hypercare

Segera setelah cutover, periode waktu ketika tim migrasi mengelola dan memantau aplikasi yang dimigrasi di cloud untuk mengatasi masalah apa pun. Biasanya, periode ini panjangnya 1-4 hari. Pada akhir periode hypercare, tim migrasi biasanya mentransfer tanggung jawab untuk aplikasi ke tim operasi cloud.

I

IAc

Lihat infrastruktur sebagai kode.

kebijakan berbasis identitas

Kebijakan yang dilampirkan pada satu atau beberapa prinsip IAM yang mendefinisikan izin mereka dalam lingkungan. AWS Cloud

aplikasi idle

Aplikasi yang memiliki penggunaan CPU dan memori rata-rata antara 5 dan 20 persen selama periode 90 hari. Dalam proyek migrasi, adalah umum untuk menghentikan aplikasi ini atau mempertahankannya di tempat.

IIoT

Lihat Internet of Things industri.

infrastruktur yang tidak dapat diubah

Model yang menyebarkan infrastruktur baru untuk beban kerja produksi alih-alih memperbarui, menambal, atau memodifikasi infrastruktur yang ada. Infrastruktur yang tidak dapat diubah secara inheren lebih konsisten, andal, dan dapat diprediksi daripada infrastruktur yang dapat berubah. Untuk informasi selengkapnya, lihat praktik terbaik Deploy using immutable infrastructure di AWS Well-Architected Framework.

masuk (masuknya) VPC

Dalam arsitektur AWS multi-akun, VPC yang menerima, memeriksa, dan merutekan koneksi jaringan dari luar aplikasi. Arsitektur Referensi AWS Keamanan merekomendasikan pengaturan akun Jaringan Anda dengan inbound, outbound, dan inspeksi VPCs untuk melindungi antarmuka dua arah antara aplikasi Anda dan internet yang lebih luas.

migrasi inkremental

Strategi cutover di mana Anda memigrasikan aplikasi Anda dalam bagian-bagian kecil alih-alih melakukan satu cutover penuh. Misalnya, Anda mungkin hanya memindahkan beberapa layanan mikro atau pengguna ke sistem baru pada awalnya. Setelah Anda memverifikasi bahwa semuanya berfungsi dengan baik, Anda dapat secara bertahap memindahkan layanan mikro atau pengguna tambahan hingga Anda dapat menonaktifkan sistem lama Anda. Strategi ini mengurangi risiko yang terkait dengan migrasi besar.

Industri 4.0

Sebuah istilah yang diperkenalkan oleh Klaus Schwab pada tahun 2016 untuk merujuk pada modernisasi proses manufaktur melalui kemajuan dalam konektivitas, data real-time, otomatisasi, analitik, dan AI/ML.

infrastruktur

Semua sumber daya dan aset yang terkandung dalam lingkungan aplikasi.

infrastruktur sebagai kode (IAc)

Proses penyediaan dan pengelolaan infrastruktur aplikasi melalui satu set file konfigurasi. IAc dirancang untuk membantu Anda memusatkan manajemen infrastruktur, menstandarisasi sumber daya, dan menskalakan dengan cepat sehingga lingkungan baru dapat diulang, andal, dan konsisten.

Internet of Things industri (IIoT)

Penggunaan sensor dan perangkat yang terhubung ke internet di sektor industri, seperti manufaktur, energi, otomotif, perawatan kesehatan, ilmu kehidupan, dan pertanian. Untuk informasi lebih lanjut, lihat Membangun strategi transformasi digital Internet of Things (IIoT) industri.

inspeksi VPC

Dalam arsitektur AWS multi-akun, VPC terpusat yang mengelola inspeksi lalu lintas jaringan antara VPCs (dalam yang sama atau berbeda Wilayah AWS), internet, dan jaringan lokal. Arsitektur Referensi AWS Keamanan merekomendasikan pengaturan akun Jaringan Anda dengan inbound, outbound, dan inspeksi VPCs untuk melindungi antarmuka dua arah antara aplikasi Anda dan internet yang lebih luas.

Internet of Things (IoT)

Jaringan objek fisik yang terhubung dengan sensor atau prosesor tertanam yang berkomunikasi dengan perangkat dan sistem lain melalui internet atau melalui jaringan komunikasi lokal. Untuk informasi selengkapnya, lihat Apa itu IoT?

interpretabilitas

Karakteristik model pembelajaran mesin yang menggambarkan sejauh mana manusia dapat memahami bagaimana prediksi model bergantung pada inputnya. Untuk informasi lebih lanjut, lihat Interpretabilitas model pembelajaran mesin dengan. AWS

IoT

Lihat Internet of Things.

Perpustakaan informasi TI (ITIL)

Serangkaian praktik terbaik untuk memberikan layanan TI dan menyelaraskan layanan ini dengan persyaratan bisnis. ITIL menyediakan dasar untuk ITSM.

Manajemen layanan TI (ITSM)

Kegiatan yang terkait dengan merancang, menerapkan, mengelola, dan mendukung layanan TI untuk suatu organisasi. Untuk informasi tentang mengintegrasikan operasi cloud dengan alat ITSM, lihat panduan integrasi operasi.

ITIL

Lihat perpustakaan informasi TI.

ITSM

Lihat manajemen layanan TI.

L

kontrol akses berbasis label (LBAC)

Implementasi kontrol akses wajib (MAC) di mana pengguna dan data itu sendiri masing-masing secara eksplisit diberi nilai label keamanan. Persimpangan antara label keamanan pengguna dan label keamanan data menentukan baris dan kolom mana yang dapat dilihat oleh pengguna.

landing zone

Landing zone adalah AWS lingkungan multi-akun yang dirancang dengan baik yang dapat diskalakan dan aman. Ini adalah titik awal dari mana organisasi Anda dapat dengan cepat meluncurkan dan menyebarkan beban kerja dan aplikasi dengan percaya diri dalam lingkungan keamanan dan infrastruktur mereka. Untuk informasi selengkapnya tentang zona pendaratan, lihat Menyiapkan lingkungan multi-akun AWS yang aman dan dapat diskalakan.

model bahasa besar (LLM)

Model AI pembelajaran mendalam yang dilatih sebelumnya pada sejumlah besar data. LLM dapat melakukan beberapa tugas, seperti menjawab pertanyaan, meringkas dokumen, menerjemahkan teks ke dalam bahasa lain, dan menyelesaikan kalimat. Untuk informasi lebih lanjut, lihat Apa itu LLMs.

migrasi besar

Migrasi 300 atau lebih server.

LBAC

Lihat kontrol akses berbasis label.

hak istimewa paling sedikit

Praktik keamanan terbaik untuk memberikan izin minimum yang diperlukan untuk melakukan tugas. Untuk informasi selengkapnya, lihat Menerapkan izin hak istimewa terkecil dalam dokumentasi IAM.

angkat dan geser

Lihat 7 Rs.

sistem endian kecil

Sebuah sistem yang menyimpan byte paling tidak signifikan terlebih dahulu. Lihat juga endianness.

LLM

Lihat model bahasa besar.

lingkungan yang lebih rendah

Lihat lingkungan.

M

pembelajaran mesin (ML)

Jenis kecerdasan buatan yang menggunakan algoritma dan teknik untuk pengenalan pola dan pembelajaran. ML menganalisis dan belajar dari data yang direkam, seperti data Internet of Things (IoT), untuk menghasilkan model statistik berdasarkan pola. Untuk informasi selengkapnya, lihat Machine Learning.

cabang utama

Lihat cabang.

malware

Perangkat lunak yang dirancang untuk membahayakan keamanan atau privasi komputer. Malware dapat mengganggu sistem komputer, membocorkan informasi sensitif, atau mendapatkan akses yang tidak sah. Contoh malware termasuk virus, worm, ransomware, Trojan horse, spyware, dan keyloggers.

layanan terkelola

Layanan AWS yang AWS mengoperasikan lapisan infrastruktur, sistem operasi, dan platform, dan Anda mengakses titik akhir untuk menyimpan dan mengambil data. Amazon Simple Storage Service (Amazon S3) dan Amazon DynamoDB adalah contoh layanan terkelola. Ini juga dikenal sebagai layanan abstrak.

sistem eksekusi manufaktur (MES)

Sistem perangkat lunak untuk melacak, memantau, mendokumentasikan, dan mengendalikan proses produksi yang mengubah bahan baku menjadi produk jadi di lantai toko.

PETA

Lihat Program Percepatan Migrasi.

mekanisme

Proses lengkap di mana Anda membuat alat, mendorong adopsi alat, dan kemudian memeriksa hasilnya untuk melakukan penyesuaian. Mekanisme adalah siklus yang memperkuat dan meningkatkan dirinya sendiri saat beroperasi. Untuk informasi lebih lanjut, lihat Membangun mekanisme di AWS Well-Architected Framework.

akun anggota

Semua Akun AWS selain akun manajemen yang merupakan bagian dari organisasi di AWS Organizations. Akun dapat menjadi anggota dari hanya satu organisasi pada suatu waktu.

MES

Lihat sistem eksekusi manufaktur.

Transportasi Telemetri Antrian Pesan (MQTT)

Protokol komunikasi ringan machine-to-machine (M2M), berdasarkan pola terbitkan/berlangganan, untuk perangkat IoT yang dibatasi sumber daya.

layanan mikro

Layanan kecil dan independen yang berkomunikasi dengan jelas APIs dan biasanya dimiliki oleh tim kecil yang mandiri. Misalnya, sistem asuransi mungkin mencakup layanan mikro yang memetakan kemampuan bisnis, seperti penjualan atau pemasaran, atau subdomain, seperti pembelian, klaim, atau analitik. Manfaat layanan mikro termasuk kelincahan, penskalaan yang fleksibel, penyebaran yang mudah, kode yang dapat digunakan kembali, dan ketahanan. Untuk informasi selengkapnya, lihat Mengintegrasikan layanan mikro dengan menggunakan layanan tanpa AWS server.

arsitektur microservices

Pendekatan untuk membangun aplikasi dengan komponen independen yang menjalankan setiap proses aplikasi sebagai layanan mikro. Layanan mikro ini berkomunikasi melalui antarmuka yang terdefinisi dengan baik dengan menggunakan ringan. APIs Setiap layanan mikro dalam arsitektur ini dapat diperbarui, digunakan, dan diskalakan untuk memenuhi permintaan fungsi tertentu dari suatu aplikasi. Untuk informasi selengkapnya, lihat Menerapkan layanan mikro di AWS.

Program Percepatan Migrasi (MAP)

AWS Program yang menyediakan dukungan konsultasi, pelatihan, dan layanan untuk membantu organisasi membangun fondasi operasional yang kuat untuk pindah ke cloud, dan untuk membantu mengimbangi biaya awal migrasi. MAP mencakup metodologi migrasi untuk mengeksekusi migrasi lama dengan cara metodis dan seperangkat alat untuk mengotomatisasi dan mempercepat skenario migrasi umum.

migrasi dalam skala

Proses memindahkan sebagian besar portofolio aplikasi ke cloud dalam gelombang, dengan lebih banyak aplikasi bergerak pada tingkat yang lebih cepat di setiap gelombang. Fase ini menggunakan praktik dan pelajaran terbaik dari fase sebelumnya untuk mengimplementasikan pabrik migrasi tim, alat, dan proses untuk merampingkan migrasi beban kerja melalui otomatisasi dan pengiriman tangkas. Ini adalah fase ketiga dari strategi AWS migrasi.

pabrik migrasi

Tim lintas fungsi yang merampingkan migrasi beban kerja melalui pendekatan otomatis dan gesit. Tim pabrik migrasi biasanya mencakup operasi, analis dan pemilik bisnis, insinyur migrasi, pengembang, dan DevOps profesional yang bekerja di sprint. Antara 20 dan 50 persen portofolio aplikasi perusahaan terdiri dari pola berulang yang dapat dioptimalkan dengan pendekatan pabrik. Untuk informasi selengkapnya, lihat diskusi tentang pabrik migrasi dan panduan Pabrik Migrasi Cloud di kumpulan konten ini.

metadata migrasi

Informasi tentang aplikasi dan server yang diperlukan untuk menyelesaikan migrasi. Setiap pola migrasi memerlukan satu set metadata migrasi yang berbeda. Contoh metadata migrasi termasuk subnet target, grup keamanan, dan akun. AWS

pola migrasi

Tugas migrasi berulang yang merinci strategi migrasi, tujuan migrasi, dan aplikasi atau layanan migrasi yang digunakan. Contoh: Rehost migrasi ke Amazon EC2 dengan Layanan Migrasi AWS Aplikasi.

Penilaian Portofolio Migrasi (MPA)

Alat online yang menyediakan informasi untuk memvalidasi kasus bisnis untuk bermigrasi ke. AWS Cloud MPA menyediakan penilaian portofolio terperinci (ukuran kanan server, harga, perbandingan TCO, analisis biaya migrasi) serta perencanaan migrasi (analisis data aplikasi dan pengumpulan data, pengelompokan aplikasi, prioritas migrasi, dan perencanaan gelombang). Alat MPA (memerlukan login) tersedia gratis untuk semua AWS konsultan dan konsultan APN Partner.

Penilaian Kesiapan Migrasi (MRA)

Proses mendapatkan wawasan tentang status kesiapan cloud organisasi, mengidentifikasi kekuatan dan kelemahan, dan membangun rencana aksi untuk menutup kesenjangan yang diidentifikasi, menggunakan CAF. AWS Untuk informasi selengkapnya, lihat panduan kesiapan migrasi. MRA adalah tahap pertama dari strategi AWS migrasi.

strategi migrasi

Pendekatan yang digunakan untuk memigrasikan beban kerja ke file. AWS Cloud Untuk informasi lebih lanjut, lihat entri 7 Rs di glosarium ini dan lihat Memobilisasi organisasi Anda untuk mempercepat migrasi skala besar.

ML

Lihat pembelajaran mesin.

modernisasi

Mengubah aplikasi usang (warisan atau monolitik) dan infrastrukturnya menjadi sistem yang gesit, elastis, dan sangat tersedia di cloud untuk mengurangi biaya, mendapatkan efisiensi, dan memanfaatkan inovasi. Untuk informasi selengkapnya, lihat Strategi untuk memodernisasi aplikasi di. AWS Cloud

penilaian kesiapan modernisasi

Evaluasi yang membantu menentukan kesiapan modernisasi aplikasi organisasi; mengidentifikasi manfaat, risiko, dan dependensi; dan menentukan seberapa baik organisasi dapat mendukung keadaan masa depan aplikasi tersebut. Hasil penilaian adalah cetak biru arsitektur target, peta jalan yang merinci fase pengembangan dan tonggak untuk proses modernisasi, dan rencana aksi untuk mengatasi kesenjangan yang diidentifikasi. Untuk informasi lebih lanjut, lihat Mengevaluasi kesiapan modernisasi untuk aplikasi di. AWS Cloud

aplikasi monolitik (monolit)

Aplikasi yang berjalan sebagai layanan tunggal dengan proses yang digabungkan secara ketat. Aplikasi monolitik memiliki beberapa kelemahan. Jika satu fitur aplikasi mengalami lonjakan permintaan, seluruh arsitektur harus diskalakan. Menambahkan atau meningkatkan fitur aplikasi monolitik juga menjadi lebih kompleks ketika basis kode tumbuh. Untuk mengatasi masalah ini, Anda dapat menggunakan arsitektur microservices. Untuk informasi lebih lanjut, lihat Menguraikan monolit menjadi layanan mikro.

MPA

Lihat Penilaian Portofolio Migrasi.

MQTT

Lihat Transportasi Telemetri Antrian Pesan.

klasifikasi multiclass

Sebuah proses yang membantu menghasilkan prediksi untuk beberapa kelas (memprediksi satu dari lebih dari dua hasil). Misalnya, model ML mungkin bertanya “Apakah produk ini buku, mobil, atau telepon?” atau “Kategori produk mana yang paling menarik bagi pelanggan ini?”

infrastruktur yang bisa berubah

Model yang memperbarui dan memodifikasi infrastruktur yang ada untuk beban kerja produksi. Untuk meningkatkan konsistensi, keandalan, dan prediktabilitas, AWS Well-Architected Framework merekomendasikan penggunaan infrastruktur yang tidak dapat diubah sebagai praktik terbaik.

O

OAC

Lihat kontrol akses asal.

OAI

Lihat identitas akses asal.

OCM

Lihat manajemen perubahan organisasi.

migrasi offline

Metode migrasi di mana beban kerja sumber diturunkan selama proses migrasi. Metode ini melibatkan waktu henti yang diperpanjang dan biasanya digunakan untuk beban kerja kecil dan tidak kritis.

OI

Lihat integrasi operasi.

OLA

Lihat perjanjian tingkat operasional.

migrasi online

Metode migrasi di mana beban kerja sumber disalin ke sistem target tanpa diambil offline. Aplikasi yang terhubung ke beban kerja dapat terus berfungsi selama migrasi. Metode ini melibatkan waktu henti nol hingga minimal dan biasanya digunakan untuk beban kerja produksi yang kritis.

OPC-UA

Lihat Komunikasi Proses Terbuka - Arsitektur Terpadu.

Komunikasi Proses Terbuka - Arsitektur Terpadu (OPC-UA)

Protokol komunikasi machine-to-machine (M2M) untuk otomasi industri. OPC-UA menyediakan standar interoperabilitas dengan enkripsi data, otentikasi, dan skema otorisasi.

perjanjian tingkat operasional (OLA)

Perjanjian yang menjelaskan apa yang dijanjikan kelompok TI fungsional untuk diberikan satu sama lain, untuk mendukung perjanjian tingkat layanan (SLA).

Tinjauan Kesiapan Operasional (ORR)

Daftar pertanyaan dan praktik terbaik terkait yang membantu Anda memahami, mengevaluasi, mencegah, atau mengurangi ruang lingkup insiden dan kemungkinan kegagalan. Untuk informasi lebih lanjut, lihat Ulasan Kesiapan Operasional (ORR) dalam Kerangka Kerja Well-Architected AWS .

teknologi operasional (OT)

Sistem perangkat keras dan perangkat lunak yang bekerja dengan lingkungan fisik untuk mengendalikan operasi industri, peralatan, dan infrastruktur. Di bidang manufaktur, integrasi sistem OT dan teknologi informasi (TI) adalah fokus utama untuk transformasi Industri 4.0.

integrasi operasi (OI)

Proses modernisasi operasi di cloud, yang melibatkan perencanaan kesiapan, otomatisasi, dan integrasi. Untuk informasi selengkapnya, lihat panduan integrasi operasi.

jejak organisasi

Jejak yang dibuat oleh AWS CloudTrail itu mencatat semua peristiwa untuk semua Akun AWS dalam organisasi di AWS Organizations. Jejak ini dibuat di setiap Akun AWS bagian organisasi dan melacak aktivitas di setiap akun. Untuk informasi selengkapnya, lihat Membuat jejak untuk organisasi dalam CloudTrail dokumentasi.

manajemen perubahan organisasi (OCM)

Kerangka kerja untuk mengelola transformasi bisnis utama yang mengganggu dari perspektif orang, budaya, dan kepemimpinan. OCM membantu organisasi mempersiapkan, dan transisi ke, sistem dan strategi baru dengan mempercepat adopsi perubahan, mengatasi masalah transisi, dan mendorong perubahan budaya dan organisasi. Dalam strategi AWS migrasi, kerangka kerja ini disebut percepatan orang, karena kecepatan perubahan yang diperlukan dalam proyek adopsi cloud. Untuk informasi lebih lanjut, lihat panduan OCM.

kontrol akses asal (OAC)

Di CloudFront, opsi yang disempurnakan untuk membatasi akses untuk mengamankan konten Amazon Simple Storage Service (Amazon S3) Anda. OAC mendukung semua bucket S3 di semua Wilayah AWS, enkripsi sisi server dengan AWS KMS (SSE-KMS), dan dinamis dan permintaan ke bucket S3. `PUT` `DELETE`

identitas akses asal (OAI)

Di CloudFront, opsi untuk membatasi akses untuk mengamankan konten Amazon S3 Anda. Saat Anda menggunakan OAI, CloudFront buat prinsipal yang dapat diautentikasi oleh Amazon S3. Prinsipal yang diautentikasi dapat mengakses konten dalam bucket S3 hanya melalui distribusi tertentu. CloudFront Lihat juga OAC, yang menyediakan kontrol akses yang lebih terperinci dan ditingkatkan.

ORR

Lihat tinjauan kesiapan operasional.

OT

Lihat teknologi operasional.

keluar (jalan keluar) VPC

Dalam arsitektur AWS multi-akun, VPC yang menangani koneksi jaringan yang dimulai dari dalam aplikasi. Arsitektur Referensi AWS Keamanan merekomendasikan pengaturan akun Jaringan Anda dengan inbound, outbound, dan inspeksi VPCs untuk melindungi antarmuka dua arah antara aplikasi Anda dan internet yang lebih luas.

P

batas izin

Kebijakan manajemen IAM yang dilampirkan pada prinsipal IAM untuk menetapkan izin maksimum yang dapat dimiliki pengguna atau peran. Untuk informasi selengkapnya, lihat Batas izin dalam dokumentasi IAM.

Informasi Identifikasi Pribadi (PII)

Informasi yang, jika dilihat secara langsung atau dipasangkan dengan data terkait lainnya, dapat digunakan untuk menyimpulkan identitas individu secara wajar. Contoh PII termasuk nama, alamat, dan informasi kontak.

PII

Lihat informasi yang dapat diidentifikasi secara pribadi.

buku pedoman

Serangkaian langkah yang telah ditentukan sebelumnya yang menangkap pekerjaan yang terkait dengan migrasi, seperti mengirimkan fungsi operasi inti di cloud. Buku pedoman dapat berupa skrip, runbook otomatis, atau ringkasan proses atau langkah-langkah yang diperlukan untuk mengoperasikan lingkungan modern Anda.

PLC

Lihat pengontrol logika yang dapat diprogram.

PLM

Lihat manajemen siklus hidup produk.

kebijakan

Objek yang dapat menentukan izin (lihat kebijakan berbasis identitas), menentukan kondisi akses (lihat kebijakan berbasis sumber daya), atau menentukan izin maksimum untuk semua akun di organisasi (lihat kebijakan kontrol layanan). AWS Organizations

ketekunan poliglot

Secara independen memilih teknologi penyimpanan data microservice berdasarkan pola akses data dan persyaratan lainnya. Jika layanan mikro Anda memiliki teknologi penyimpanan data yang sama, mereka dapat menghadapi tantangan implementasi atau mengalami kinerja yang buruk. Layanan mikro lebih mudah diimplementasikan dan mencapai kinerja dan skalabilitas yang lebih baik jika mereka menggunakan penyimpanan data yang paling sesuai dengan kebutuhan mereka. Untuk informasi selengkapnya, lihat Mengaktifkan persistensi data di layanan mikro.

penilaian portofolio

Proses menemukan, menganalisis, dan memprioritaskan portofolio aplikasi untuk merencanakan migrasi. Untuk informasi selengkapnya, lihat Mengevaluasi kesiapan migrasi.

predikat

Kondisi kueri yang mengembalikan `true` atau`false`, biasanya terletak di `WHERE` klausa.

predikat pushdown

Teknik optimasi kueri database yang menyaring data dalam kueri sebelum transfer. Ini mengurangi jumlah data yang harus diambil dan diproses dari database relasional, dan meningkatkan kinerja kueri.

kontrol preventif

Kontrol keamanan yang dirancang untuk mencegah suatu peristiwa terjadi. Kontrol ini adalah garis pertahanan pertama untuk membantu mencegah akses tidak sah atau perubahan yang tidak diinginkan ke jaringan Anda. Untuk informasi selengkapnya, lihat Kontrol pencegahan dalam Menerapkan kontrol keamanan pada. AWS

principal

Entitas AWS yang dapat melakukan tindakan dan mengakses sumber daya. Entitas ini biasanya merupakan pengguna root untuk Akun AWS, peran IAM, atau pengguna. Untuk informasi selengkapnya, lihat Prinsip dalam istilah dan konsep Peran dalam dokumentasi IAM.

privasi berdasarkan desain

Pendekatan rekayasa sistem yang memperhitungkan privasi melalui seluruh proses pengembangan.

zona yang dihosting pribadi

Container yang menyimpan informasi tentang bagaimana Anda ingin Amazon Route 53 merespons kueri DNS untuk domain dan subdomainnya dalam satu atau lebih. VPCs Untuk informasi selengkapnya, lihat Bekerja dengan zona yang dihosting pribadi di dokumentasi Route 53.

kontrol proaktif

Kontrol keamanan yang dirancang untuk mencegah penyebaran sumber daya yang tidak sesuai. Kontrol ini memindai sumber daya sebelum disediakan. Jika sumber daya tidak sesuai dengan kontrol, maka itu tidak disediakan. Untuk informasi selengkapnya, lihat panduan referensi Kontrol dalam AWS Control Tower dokumentasi dan lihat Kontrol proaktif dalam Menerapkan kontrol keamanan pada AWS.

manajemen siklus hidup produk (PLM)

Manajemen data dan proses untuk suatu produk di seluruh siklus hidupnya, mulai dari desain, pengembangan, dan peluncuran, melalui pertumbuhan dan kematangan, hingga penurunan dan penghapusan.

lingkungan produksi

Lihat lingkungan.

pengontrol logika yang dapat diprogram (PLC)

Di bidang manufaktur, komputer yang sangat andal dan mudah beradaptasi yang memantau mesin dan mengotomatiskan proses manufaktur.

rantai cepat

Menggunakan output dari satu prompt LLM sebagai input untuk prompt berikutnya untuk menghasilkan respons yang lebih baik. Teknik ini digunakan untuk memecah tugas yang kompleks menjadi subtugas, atau untuk secara iteratif memperbaiki atau memperluas respons awal. Ini membantu meningkatkan akurasi dan relevansi respons model dan memungkinkan hasil yang lebih terperinci dan dipersonalisasi.

pseudonimisasi

Proses penggantian pengenal pribadi dalam kumpulan data dengan nilai placeholder. Pseudonimisasi dapat membantu melindungi privasi pribadi. Data pseudonim masih dianggap sebagai data pribadi.

publish/subscribe (pub/sub)

Pola yang memungkinkan komunikasi asinkron antara layanan mikro untuk meningkatkan skalabilitas dan daya tanggap. Misalnya, dalam MES berbasis layanan mikro, layanan mikro dapat mempublikasikan pesan peristiwa ke saluran yang dapat berlangganan layanan mikro lainnya. Sistem dapat menambahkan layanan mikro baru tanpa mengubah layanan penerbitan.

Q

rencana kueri

Serangkaian langkah, seperti instruksi, yang digunakan untuk mengakses data dalam sistem database relasional SQL.

regresi rencana kueri

Ketika pengoptimal layanan database memilih rencana yang kurang optimal daripada sebelum perubahan yang diberikan ke lingkungan database. Hal ini dapat disebabkan oleh perubahan statistik, kendala, pengaturan lingkungan, pengikatan parameter kueri, dan pembaruan ke mesin database.

R

Matriks RACI

Lihat bertanggung jawab, akuntabel, dikonsultasikan, diinformasikan (RACI).

LAP

Lihat Retrieval Augmented Generation.

ransomware

Perangkat lunak berbahaya yang dirancang untuk memblokir akses ke sistem komputer atau data sampai pembayaran dilakukan.

Matriks RASCI

Lihat bertanggung jawab, akuntabel, dikonsultasikan, diinformasikan (RACI).

RCAC

Lihat kontrol akses baris dan kolom.

replika baca

Salinan database yang digunakan untuk tujuan read-only. Anda dapat merutekan kueri ke replika baca untuk mengurangi beban pada database utama Anda.

arsitek ulang

Lihat 7 Rs.

tujuan titik pemulihan (RPO)

Jumlah waktu maksimum yang dapat diterima sejak titik pemulihan data terakhir. Ini menentukan apa yang dianggap sebagai kehilangan data yang dapat diterima antara titik pemulihan terakhir dan gangguan layanan.

tujuan waktu pemulihan (RTO)

Penundaan maksimum yang dapat diterima antara gangguan layanan dan pemulihan layanan.

refactor

Lihat 7 Rs.

Wilayah

Kumpulan AWS sumber daya di wilayah geografis. Masing-masing Wilayah AWS terisolasi dan independen dari yang lain untuk memberikan toleransi kesalahan, stabilitas, dan ketahanan. Untuk informasi selengkapnya, lihat Menentukan Wilayah AWS akun yang dapat digunakan.

regresi

Teknik ML yang memprediksi nilai numerik. Misalnya, untuk memecahkan masalah “Berapa harga rumah ini akan dijual?” Model ML dapat menggunakan model regresi linier untuk memprediksi harga jual rumah berdasarkan fakta yang diketahui tentang rumah (misalnya, luas persegi).

rehost

Lihat 7 Rs.

melepaskan

Dalam proses penyebaran, tindakan mempromosikan perubahan pada lingkungan produksi.

memindahkan

Lihat 7 Rs.

memplatform ulang

Lihat 7 Rs.

pembelian kembali

Lihat 7 Rs.

ketahanan

Kemampuan aplikasi untuk melawan atau pulih dari gangguan. Ketersediaan tinggi dan pemulihan bencana adalah pertimbangan umum ketika merencanakan ketahanan di. AWS Cloud Untuk informasi lebih lanjut, lihat AWS Cloud Ketahanan.

kebijakan berbasis sumber daya

Kebijakan yang dilampirkan ke sumber daya, seperti bucket Amazon S3, titik akhir, atau kunci enkripsi. Jenis kebijakan ini menentukan prinsipal mana yang diizinkan mengakses, tindakan yang didukung, dan kondisi lain yang harus dipenuhi.

matriks yang bertanggung jawab, akuntabel, dikonsultasikan, diinformasikan (RACI)

Matriks yang mendefinisikan peran dan tanggung jawab untuk semua pihak yang terlibat dalam kegiatan migrasi dan operasi cloud. Nama matriks berasal dari jenis tanggung jawab yang

didefinisikan dalam matriks: bertanggung jawab (R), akuntabel (A), dikonsultasikan (C), dan diinformasikan (I). Tipe dukungan (S) adalah opsional. Jika Anda menyertakan dukungan, matriks disebut matriks RASCI, dan jika Anda mengecualikannya, itu disebut matriks RACI.

kontrol responsif

Kontrol keamanan yang dirancang untuk mendorong remediasi efek samping atau penyimpangan dari garis dasar keamanan Anda. Untuk informasi selengkapnya, lihat Kontrol responsif dalam Menerapkan kontrol keamanan pada AWS.

melestarikan

Lihat 7 Rs.

pensiun

Lihat 7 Rs.

Retrieval Augmented Generation (RAG)

Teknologi AI generatif di mana LLM merujuk sumber data otoritatif yang berada di luar sumber data pelatihannya sebelum menghasilkan respons. Misalnya, model RAG mungkin melakukan pencarian semantik dari basis pengetahuan organisasi atau data kustom. Untuk informasi lebih lanjut, lihat Apa itu RAG.

rotasi

Proses memperbarui rahasia secara berkala untuk membuatnya lebih sulit bagi penyerang untuk mengakses kredensil.

kontrol akses baris dan kolom (RCAC)

Penggunaan ekspresi SQL dasar dan fleksibel yang telah menetapkan aturan akses. RCAC terdiri dari izin baris dan topeng kolom.

RPO

Lihat tujuan titik pemulihan.

RTO

Lihat tujuan waktu pemulihan.

buku runbook

Satu set prosedur manual atau otomatis yang diperlukan untuk melakukan tugas tertentu. Ini biasanya dibangun untuk merampingkan operasi berulang atau prosedur dengan tingkat kesalahan yang tinggi.

D

SAML 2.0

Standar terbuka yang digunakan oleh banyak penyedia identitas (IdPs). Fitur ini memungkinkan sistem masuk tunggal gabungan (SSO), sehingga pengguna dapat masuk ke AWS Management Console atau memanggil operasi AWS API tanpa Anda harus membuat pengguna di IAM untuk semua orang di organisasi Anda. Untuk informasi lebih lanjut tentang federasi berbasis SAMP 2.0, lihat Tentang federasi berbasis SAMP 2.0 dalam dokumentasi IAM.

SCADA

Lihat kontrol pengawasan dan akuisisi data.

SCP

Lihat kebijakan kontrol layanan.

Rahasia

Dalam AWS Secrets Manager, informasi rahasia atau terbatas, seperti kata sandi atau kredensil pengguna, yang Anda simpan dalam bentuk terenkripsi. Ini terdiri dari nilai rahasia dan metadata-nya. Nilai rahasia dapat berupa biner, string tunggal, atau beberapa string. Untuk informasi selengkapnya, lihat Apa yang ada di rahasia Secrets Manager? dalam dokumentasi Secrets Manager.

keamanan dengan desain

Pendekatan rekayasa sistem yang memperhitungkan keamanan melalui seluruh proses pengembangan.

kontrol keamanan

Pagar pembatas teknis atau administratif yang mencegah, mendeteksi, atau mengurangi kemampuan pelaku ancaman untuk mengeksploitasi kerentanan keamanan. Ada empat jenis kontrol keamanan utama: preventif, detektif, responsif, dan proaktif.

pengerasan keamanan

Proses mengurangi permukaan serangan untuk membuatnya lebih tahan terhadap serangan. Ini dapat mencakup tindakan seperti menghapus sumber daya yang tidak lagi diperlukan, menerapkan praktik keamanan terbaik untuk memberikan hak istimewa paling sedikit, atau menonaktifkan fitur yang tidak perlu dalam file konfigurasi.

sistem informasi keamanan dan manajemen acara (SIEM)

Alat dan layanan yang menggabungkan sistem manajemen informasi keamanan (SIM) dan manajemen acara keamanan (SEM). Sistem SIEM mengumpulkan, memantau, dan menganalisis data dari server, jaringan, perangkat, dan sumber lain untuk mendeteksi ancaman dan pelanggaran keamanan, dan untuk menghasilkan peringatan.

otomatisasi respons keamanan

Tindakan yang telah ditentukan dan diprogram yang dirancang untuk secara otomatis merespons atau memulihkan peristiwa keamanan. Otomatisasi ini berfungsi sebagai kontrol keamanan detektif atau responsif yang membantu Anda menerapkan praktik terbaik AWS keamanan. Contoh tindakan respons otomatis termasuk memodifikasi grup keamanan VPC, menambal instans EC2 Amazon, atau memutar kredensil.

enkripsi sisi server

Enkripsi data di tujuannya, oleh Layanan AWS yang menerimanya.

kebijakan kontrol layanan (SCP)

Kebijakan yang menyediakan kontrol terpusat atas izin untuk semua akun di organisasi. AWS Organizations SCPs menentukan pagar pembatas atau menetapkan batasan pada tindakan yang dapat didelegasikan oleh administrator kepada pengguna atau peran. Anda dapat menggunakan SCPs daftar izin atau daftar penolakan, untuk menentukan layanan atau tindakan mana yang diizinkan atau dilarang. Untuk informasi selengkapnya, lihat Kebijakan kontrol layanan dalam AWS Organizations dokumentasi.

titik akhir layanan

URL titik masuk untuk file Layanan AWS. Anda dapat menggunakan endpoint untuk terhubung secara terprogram ke layanan target. Untuk informasi selengkapnya, lihat Layanan AWS titik akhir di Referensi Umum AWS.

perjanjian tingkat layanan (SLA)

Perjanjian yang menjelaskan apa yang dijanjikan tim TI untuk diberikan kepada pelanggan mereka, seperti uptime dan kinerja layanan.

indikator tingkat layanan (SLI)

Pengukuran aspek kinerja layanan, seperti tingkat kesalahan, ketersediaan, atau throughputnya.

tujuan tingkat layanan (SLO)

Metrik target yang mewakili kesehatan layanan, yang diukur dengan indikator tingkat layanan.

model tanggung jawab bersama

Model yang menjelaskan tanggung jawab yang Anda bagikan AWS untuk keamanan dan kepatuhan cloud. AWS bertanggung jawab atas keamanan cloud, sedangkan Anda bertanggung jawab atas keamanan di cloud. Untuk informasi selengkapnya, lihat Model tanggung jawab bersama.

SIEM

Lihat informasi keamanan dan sistem manajemen acara.

titik kegagalan tunggal (SPOF)

Kegagalan dalam satu komponen penting dari aplikasi yang dapat mengganggu sistem.

SLA

Lihat perjanjian tingkat layanan.

SLI

Lihat indikator tingkat layanan.

SLO

Lihat tujuan tingkat layanan.

split-and-seed model

Pola untuk menskalakan dan mempercepat proyek modernisasi. Ketika fitur baru dan rilis produk didefinisikan, tim inti berpisah untuk membuat tim produk baru. Ini membantu meningkatkan kemampuan dan layanan organisasi Anda, meningkatkan produktivitas pengembang, dan

mendukung inovasi yang cepat. Untuk informasi lebih lanjut, lihat Pendekatan bertahap untuk memodernisasi aplikasi di. AWS Cloud

SPOF

Lihat satu titik kegagalan.

skema bintang

Struktur organisasi database yang menggunakan satu tabel fakta besar untuk menyimpan data transaksional atau terukur dan menggunakan satu atau lebih tabel dimensi yang lebih kecil untuk menyimpan atribut data. Struktur ini dirancang untuk digunakan dalam gudang data atau untuk tujuan intelijen bisnis.

pola ara pencekik

Pendekatan untuk memodernisasi sistem monolitik dengan menulis ulang secara bertahap dan mengganti fungsionalitas sistem sampai sistem warisan dapat dinonaktifkan. Pola ini menggunakan analogi pohon ara yang tumbuh menjadi pohon yang sudah mapan dan akhirnya mengatasi dan menggantikan inangnya. Pola ini diperkenalkan oleh Martin Fowler sebagai cara untuk mengelola risiko saat menulis ulang sistem monolitik. Untuk contoh cara menerapkan pola ini, lihat Memodernisasi layanan web Microsoft ASP.NET (ASMX) lama secara bertahap menggunakan container dan Amazon API Gateway.

subnet

Rentang alamat IP dalam VPC Anda. Subnet harus berada di Availability Zone tunggal.

kontrol pengawasan dan akuisisi data (SCADA)

Di bidang manufaktur, sistem yang menggunakan perangkat keras dan perangkat lunak untuk memantau aset fisik dan operasi produksi.

enkripsi simetris

Algoritma enkripsi yang menggunakan kunci yang sama untuk mengenkripsi dan mendekripsi data.

pengujian sintetis

Menguji sistem dengan cara yang mensimulasikan interaksi pengguna untuk mendeteksi potensi masalah atau untuk memantau kinerja. Anda dapat menggunakan Amazon CloudWatch Synthetics untuk membuat tes ini.

sistem prompt

Teknik untuk memberikan konteks, instruksi, atau pedoman ke LLM untuk mengarahkan perilakunya. Permintaan sistem membantu mengatur konteks dan menetapkan aturan untuk interaksi dengan pengguna.

T

tag

Pasangan nilai kunci yang bertindak sebagai metadata untuk mengatur sumber daya Anda. AWS Tanda dapat membantu Anda mengelola, mengidentifikasi, mengatur, dan memfilter sumber daya. Untuk informasi selengkapnya, lihat Menandai AWS sumber daya Anda.

variabel target

Nilai yang Anda coba prediksi dalam ML yang diawasi. Ini juga disebut sebagai variabel hasil. Misalnya, dalam pengaturan manufaktur, variabel target bisa menjadi cacat produk.

daftar tugas

Alat yang digunakan untuk melacak kemajuan melalui runbook. Daftar tugas berisi ikhtisar runbook dan daftar tugas umum yang harus diselesaikan. Untuk setiap tugas umum, itu termasuk perkiraan jumlah waktu yang dibutuhkan, pemilik, dan kemajuan.

lingkungan uji

Lihat lingkungan.

pelatihan

Untuk menyediakan data bagi model ML Anda untuk dipelajari. Data pelatihan harus berisi jawaban yang benar. Algoritma pembelajaran menemukan pola dalam data pelatihan yang memetakan atribut data input ke target (jawaban yang ingin Anda prediksi). Ini menghasilkan model ML yang menangkap pola-pola ini. Anda kemudian dapat menggunakan model ML untuk membuat prediksi pada data baru yang Anda tidak tahu targetnya.

gerbang transit

Hub transit jaringan yang dapat Anda gunakan untuk menghubungkan jaringan Anda VPCs dan lokal. Untuk informasi selengkapnya, lihat Apa itu gateway transit dalam AWS Transit Gateway dokumentasi.

alur kerja berbasis batang

Pendekatan di mana pengembang membangun dan menguji fitur secara lokal di cabang fitur dan kemudian menggabungkan perubahan tersebut ke cabang utama. Cabang utama kemudian dibangun untuk pengembangan, praproduksi, dan lingkungan produksi, secara berurutan.

akses tepercaya

Memberikan izin ke layanan yang Anda tentukan untuk melakukan tugas di organisasi Anda di dalam AWS Organizations dan di akunnya atas nama Anda. Layanan tepercaya menciptakan peran terkait layanan di setiap akun, ketika peran itu diperlukan, untuk melakukan tugas manajemen untuk Anda. Untuk informasi selengkapnya, lihat Menggunakan AWS Organizations dengan AWS layanan lain dalam AWS Organizations dokumentasi.

penyetelan

Untuk mengubah aspek proses pelatihan Anda untuk meningkatkan akurasi model ML. Misalnya, Anda dapat melatih model ML dengan membuat set pelabelan, menambahkan label, dan kemudian mengulangi langkah-langkah ini beberapa kali di bawah pengaturan yang berbeda untuk mengoptimalkan model.

tim dua pizza

Sebuah DevOps tim kecil yang bisa Anda beri makan dengan dua pizza. Ukuran tim dua pizza memastikan peluang terbaik untuk berkolaborasi dalam pengembangan perangkat lunak.

U

waswas

Sebuah konsep yang mengacu pada informasi yang tidak tepat, tidak lengkap, atau tidak diketahui yang dapat merusak keandalan model ML prediktif. Ada dua jenis ketidakpastian: ketidakpastian epistemik disebabkan oleh data yang terbatas dan tidak lengkap, sedangkan ketidakpastian aleatorik disebabkan oleh kebisingan dan keacakan yang melekat dalam data. Untuk informasi lebih lanjut, lihat panduan Mengukur ketidakpastian dalam sistem pembelajaran mendalam.

tugas yang tidak terdiferensiasi

Juga dikenal sebagai angkat berat, pekerjaan yang diperlukan untuk membuat dan mengoperasikan aplikasi tetapi itu tidak memberikan nilai langsung kepada pengguna akhir atau

memberikan keunggulan kompetitif. Contoh tugas yang tidak terdiferensiasi termasuk pengadaan, pemeliharaan, dan perencanaan kapasitas.

lingkungan atas

Lihat lingkungan.

V

menyedot debu

Operasi pemeliharaan database yang melibatkan pembersihan setelah pembaruan tambahan untuk merebut kembali penyimpanan dan meningkatkan kinerja.

kendali versi

Proses dan alat yang melacak perubahan, seperti perubahan kode sumber dalam repositori.

Peering VPC

Koneksi antara dua VPCs yang memungkinkan Anda untuk merutekan lalu lintas dengan menggunakan alamat IP pribadi. Untuk informasi selengkapnya, lihat Apa itu peering VPC di dokumentasi VPC Amazon.

kerentanan

Kelemahan perangkat lunak atau perangkat keras yang membahayakan keamanan sistem.

W

cache hangat

Cache buffer yang berisi data saat ini dan relevan yang sering diakses. Instance database dapat membaca dari cache buffer, yang lebih cepat daripada membaca dari memori utama atau disk.

data hangat

Data yang jarang diakses. Saat menanyakan jenis data ini, kueri yang cukup lambat biasanya dapat diterima.

fungsi jendela

Fungsi SQL yang melakukan perhitungan pada sekelompok baris yang berhubungan dengan catatan saat ini. Fungsi jendela berguna untuk memproses tugas, seperti menghitung rata-rata bergerak atau mengakses nilai baris berdasarkan posisi relatif dari baris saat ini.

beban kerja

Kumpulan sumber daya dan kode yang memberikan nilai bisnis, seperti aplikasi yang dihadapi pelanggan atau proses backend.

aliran kerja

Grup fungsional dalam proyek migrasi yang bertanggung jawab atas serangkaian tugas tertentu. Setiap alur kerja independen tetapi mendukung alur kerja lain dalam proyek. Misalnya, alur kerja portofolio bertanggung jawab untuk memprioritaskan aplikasi, perencanaan gelombang, dan mengumpulkan metadata migrasi. Alur kerja portofolio mengirimkan aset ini ke alur kerja migrasi, yang kemudian memigrasikan server dan aplikasi.

CACING

Lihat menulis sekali, baca banyak.

WQF

Lihat AWS Kerangka Kualifikasi Beban Kerja.

tulis sekali, baca banyak (WORM)

Model penyimpanan yang menulis data satu kali dan mencegah data dihapus atau dimodifikasi. Pengguna yang berwenang dapat membaca data sebanyak yang diperlukan, tetapi mereka tidak dapat mengubahnya. Infrastruktur penyimpanan data ini dianggap tidak dapat diubah.

Z

eksploitasi zero-day

Serangan, biasanya malware, yang memanfaatkan kerentanan zero-day.

kerentanan zero-day

Cacat atau kerentanan yang tak tanggung-tanggung dalam sistem produksi. Aktor ancaman dapat menggunakan jenis kerentanan ini untuk menyerang sistem. Pengembang sering menyadari kerentanan sebagai akibat dari serangan tersebut.

bisikan zero-shot

Memberikan LLM dengan instruksi untuk melakukan tugas tetapi tidak ada contoh (tembakan) yang dapat membantu membimbingnya. LLM harus menggunakan pengetahuan pra-terlatih untuk menangani tugas. Efektivitas bidikan nol tergantung pada kompleksitas tugas dan kualitas prompt. Lihat juga beberapa bidikan yang diminta.

aplikasi zombie

Aplikasi yang memiliki CPU rata-rata dan penggunaan memori di bawah 5 persen. Dalam proyek migrasi, adalah umum untuk menghentikan aplikasi ini.

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