Choisir une stratégie de branchement Git pour les environnements multi-comptes DevOps

AWS Conseils prescriptifs

AWS Conseils prescriptifs: Choisir une stratégie de branchement Git pour les environnements multi-comptes DevOps

Table of Contents

Introduction 1
Objectifs 1
CI/CD Pratiques d'utilisation 2
Comprendre les DevOps environnements 4
Environnement Sandbox 5
Accès 5
Étapes de construction 5
Étapes de déploiement 6
Attentes avant de passer à l'environnement de développement 6
Environnement de développement 6
Accès 5
Étapes de construction 5
Étapes de déploiement 6
Attentes avant de passer à l'environnement de test 7
Environnement de test 8
Accès 5
Étapes de construction 5
Étapes de déploiement 6
Attentes avant de passer à l'environnement de mise en scène 9
Environnement de mise en scène 9
Accès 5
Étapes de construction 5
Étapes de déploiement 6
Attentes avant de passer à l'environnement de production 10
Environnement de production 11
Accès 5
Étapes de construction 5
Étapes de déploiement 6
Bonnes pratiques pour le développement basé sur Git 12
Stratégies de branchement Git 14
Stratégie de branchement des troncs 14
Aperçu visuel de la stratégie Trunk 15
Stratégie Branches in a Trunk 16
Avantages et inconvénients de la stratégie Trunk 18

GitHub Stratégie de branchement des flux 21
Aperçu visuel de la stratégie GitHub Flow 22
Branches dans une stratégie GitHub Flow 23
Avantages et inconvénients de la stratégie GitHub Flow 25
Stratégie de branchement Gitflow 27
Aperçu visuel de la stratégie Gitflow 28
Branches dans une stratégie Gitflow 30
Avantages et inconvénients de la stratégie Gitflow 34
Étapes suivantes 36
Ressources 37
AWS Conseils prescriptifs 37
Autres AWS conseils 37
Autres ressources 37
Collaborateurs 39
Conception 39
Révision 39
Rédaction technique 39
Historique du document 40
..... xli

Choisir une stratégie de branchement Git pour les environnements multi-comptes DevOps

Amazon Web Services (contributeurs)

Février 2024 (historique du document)

L'adoption d'une approche basée sur le cloud et la fourniture de solutions logicielles AWS peuvent être transformatrices. Cela peut nécessiter des modifications de votre processus de cycle de vie de développement logiciel. Généralement, plusieurs Comptes AWS sont utilisés au cours du processus de développement dans le AWS Cloud. Pour réussir, il est essentiel de choisir une stratégie de branchement Git compatible à associer à vos DevOps processus. Le choix de la stratégie de branchement Git adaptée à votre organisation vous permet de communiquer de manière concise les DevOps normes et les meilleures pratiques au sein des équipes de développement. Le branchement Git peut être simple dans un environnement unique, mais il peut être source de confusion lorsqu'il est appliqué à plusieurs environnements, tels que les environnements de sandbox, de développement, de test, de préparation et de production. Le fait de disposer de plusieurs environnements augmente la complexité de la DevOps mise en œuvre.

Ce guide fournit des diagrammes visuels des stratégies de branchement de Git qui montrent comment une organisation peut implémenter un processus multi-comptes DevOps . Les guides visuels aident les équipes à comprendre comment intégrer leurs stratégies de branchement Git à leurs DevOps pratiques. L'utilisation d'un modèle de branchement standard, tel que GitHub Gitflow, Flow ou Trunk, pour gérer le référentiel de code source aide les équipes de développement à aligner leur travail. Ces équipes peuvent également utiliser les ressources de formation Git standard sur Internet pour comprendre et mettre en œuvre ces modèles et stratégies.

Pour connaître les DevOps meilleures pratiques en la matière AWS, consultez le DevOpsguide publié dans AWS Well-Architected. Au cours de votre lecture de ce guide, faites preuve de diligence raisonnable pour sélectionner la stratégie de succursale adaptée à votre organisation. Certaines stratégies peuvent mieux correspondre à votre cas d'utilisation que d'autres.

Objectifs

Ce guide fait partie d'une série de documentation sur le choix et la mise en œuvre de stratégies de création de DevOps succursales pour les organisations qui en ont plusieurs Comptes AWS. Cette

série est conçue pour vous aider à appliquer la stratégie qui répond le mieux à vos exigences, à vos objectifs et à vos meilleures pratiques dès le départ, afin de rationaliser votre expérience dans le AWS Cloud. Ce guide ne contient pas de scripts DevOps exécutables car ils varient en fonction du moteur d'intégration continue et de livraison continue (CI/CD) et des cadres technologiques utilisés par votre organisation.

Ce guide explique les différences entre trois stratégies de branchement Git courantes : GitHub Flow, Gitflow et Trunk. Les recommandations de ce guide aident les équipes à identifier une stratégie de succursale conforme à leurs objectifs organisationnels. Après avoir lu ce guide, vous devriez être en mesure de choisir une stratégie de succursale pour votre organisation. Après avoir choisi une stratégie, vous pouvez utiliser l'un des modèles suivants pour vous aider à mettre en œuvre cette stratégie avec vos équipes de développement :

- Mettre en œuvre une stratégie de branchement de type Trunk pour les environnements multi-comptes DevOps
- Mettre en œuvre une stratégie GitHub de branchement Flow pour les environnements multi-comptes DevOps
- Mettre en œuvre une stratégie de branchement Gitflow pour les environnements multi-comptes DevOps

Il est important de noter que ce qui fonctionne pour une organisation, une équipe ou un projet peut ne pas convenir à d'autres. Le choix entre les stratégies de branchement Git dépend de divers facteurs, tels que la taille de l'équipe, les exigences du projet et l'équilibre souhaité entre la collaboration, la fréquence d'intégration et la gestion des versions.

CI/CD Pratiques d'utilisation

AWS vous recommande de mettre en œuvre l'intégration et la livraison continues (les 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/CD pipelines fournissent également une gouvernance et des garde-fous aux équipes de développement). Ils appliquent la cohérence, les normes, les meilleures pratiques et les niveaux d'acceptation minimaux pour l'acceptation et le

déploiement des fonctionnalités. Pour plus d'informations, voir Pratiquer l'intégration continue et la livraison continue sur AWS.

Toutes les stratégies de branchement décrites dans ce guide sont bien adaptées pour 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 établir des garde-fous et des mécanismes pour les équipes de développement afin d'empêcher les développeurs de contourner intentionnellement ou non le processus défini.

AWS propose une suite de services de développement conçus pour vous aider à créer des pipelines CI/CD. Par exemple, AWS CodePipelineil s'agit d'un service de livraison continue entièrement géré qui vous aide à automatiser vos pipelines de publication pour des mises à jour rapides et fiables des applications et de l'infrastructure. AWS CodeBuildcompile le code source, exécute des tests et produit des ready-to-deploy progiciels. Pour plus d'informations, consultez la section Outils de développement sur AWS.

Comprendre les DevOps environnements

Pour comprendre les stratégies de branchement, vous devez comprendre l'objectif et les activités de chaque environnement. La mise en place de plusieurs environnements vous permet de séparer les activités de développement en plusieurs étapes, de surveiller ces activités et d'empêcher le lancement involontaire de fonctionnalités non approuvées. Vous pouvez en avoir un ou plusieurs Comptes AWS dans chaque environnement.

La plupart des organisations ont défini plusieurs environnements à utiliser. Cependant, le nombre d'environnements peut varier en fonction des organisations et des politiques de développement logiciel. Cette série de documentation part du principe que vous disposez des cinq environnements courants suivants qui couvrent votre pipeline de développement, bien qu'ils puissent porter des noms différents :

- Sandbox : environnement dans lequel les développeurs écrivent du code, commettent des erreurs et réalisent des travaux de validation de concept.
- Développement : environnement dans lequel les développeurs intègrent leur code pour s'assurer qu'il fonctionne comme une seule et même application cohérente.
- Tests : environnement dans lequel les équipes d'assurance qualité ou les tests d'acceptation ont lieu. Les équipes effectuent souvent des tests de performance ou d'intégration dans cet environnement.
- Stage : environnement de préproduction dans lequel vous confirmez que le code et l'infrastructure fonctionnent comme prévu dans des circonstances équivalentes à celles de la production. Cet environnement est configuré pour être aussi similaire que possible à l'environnement de production.
- Production : environnement qui gère le trafic provenant de vos utilisateurs finaux et de vos clients.

Cette section décrit chaque environnement en détail. Il décrit également les étapes de création, les étapes de déploiement et les critères de sortie pour chaque environnement afin que vous puissiez passer au suivant. L'image suivante montre ces environnements dans l'ordre.

Rubriques de cette section :

- Environnement Sandbox
- Environnement de développement
- Environnement de test
- Environnement de mise en scène
- Environnement de production

Environnement Sandbox

L'environnement sandbox est l'endroit où les développeurs écrivent du code, commettent des erreurs et réalisent des travaux de validation de concept. Vous pouvez effectuer un déploiement dans un environnement sandbox à partir d'un poste de travail local ou via un script sur un poste de travail local.

Accès

Les développeurs doivent avoir un accès complet à l'environnement sandbox.

Étapes de construction

Les développeurs exécutent manuellement le build sur leurs postes de travail locaux lorsqu'ils sont prêts à déployer des modifications dans l'environnement sandbox.

1. Utilisez git-secrets (GitHub) pour rechercher des informations sensibles
2. Lint le code source
3. Compilez et compilez le code source, le cas échéant
4. Réaliser des tests unitaires
5. Réaliser une analyse de la couverture du code
6. Effectuez une analyse de code statique
7. Construire une infrastructure sous forme de code (IaC)
8. Réaliser une analyse de sécurité IaC
9. Extraire des licences open source
10. Publier des artefacts de construction

Étapes de déploiement

Si vous utilisez les modèles Gitflow ou Trunk, les étapes de déploiement démarrent automatiquement lorsqu'une `feature` branche est correctement créée dans l'environnement sandbox. Si vous utilisez le modèle GitHub Flow, vous devez exécuter manuellement les étapes de déploiement suivantes. Les étapes de déploiement dans l'environnement sandbox sont les suivantes :

1. Télécharger les artefacts publiés
2. Effectuer le versionnement de la base de données
3. Effectuer le déploiement d'iAc
4. Réaliser des tests d'intégration

Attentes avant de passer à l'environnement de développement

- Création réussie de la `feature` branche dans l'environnement sandbox
- Un développeur a déployé et testé manuellement la fonctionnalité dans l'environnement sandbox

Environnement de développement

L'environnement de développement est l'endroit où les développeurs intègrent leur code afin de garantir qu'il fonctionne comme une seule application cohérente. Dans Gitflow, l'environnement de développement contient les dernières fonctionnalités incluses par demande de fusion et est prêt à être publié. Dans les stratégies GitHub Flow et Trunk, l'environnement de développement est considéré comme un environnement de test, et la base de code peut être instable et ne pas convenir au déploiement en production.

Accès

Attribuez des autorisations selon le principe du moindre privilège. Le moindre privilège est la bonne pratique de sécurité qui consiste à accorder les autorisations minimales nécessaires à l'exécution d'une tâche. Les développeurs devraient avoir moins accès à l'environnement de développement qu'à l'environnement sandbox.

Étapes de construction

La création d'une demande de fusion vers la `develop` branche (Gitflow) ou la `main` branche (Trunk ou GitHub Flow) lance automatiquement la construction.

1. Utilisez git-secrets (GitHub) pour rechercher des informations sensibles
2. Lint le code source
3. Compilez et compilez le code source, le cas échéant
4. Réaliser des tests unitaires
5. Réaliser une analyse de la couverture du code
6. Effectuez une analyse de code statique
7. Construisez iAc
8. Réaliser une analyse de sécurité IaC
9. Extraire des licences open source

Étapes de déploiement

Si vous utilisez le modèle Gitflow, les étapes de déploiement démarrent automatiquement lorsqu'une `develop` branche est correctement créée dans l'environnement de développement. Si vous utilisez le modèle GitHub Flow ou le modèle Trunk, les étapes de déploiement démarrent automatiquement lorsqu'une demande de fusion est créée pour la `main` branche. Les étapes de déploiement dans l'environnement de développement sont les suivantes :

1. Téléchargez les artefacts publiés à partir des étapes de construction
2. Effectuer le versionnement de la base de données
3. Effectuer le déploiement d'iAc
4. Réaliser des tests d'intégration

Attentes avant de passer à l'environnement de test

- Création et déploiement réussis de la `develop` branche (Gitflow) ou de la `main` branche (Trunk ou GitHub Flow) dans l'environnement de développement
- Les tests unitaires passent à 100 %

- Construction réussie d'iAc
- Les artefacts de déploiement ont été créés avec succès
- Un développeur a effectué une vérification manuelle pour confirmer que la fonctionnalité fonctionne comme prévu

Environnement de test

Le personnel chargé de l'assurance qualité (AQ) utilise l'environnement de test pour valider les fonctionnalités. Ils approuvent les modifications une fois les tests terminés. Après approbation, la branche passe à l'environnement suivant, celui du stage. Dans Gitflow, cet environnement et les autres environnements supérieurs ne sont disponibles que pour le déploiement à partir de `release` succursales. Une `release` branche est basée sur une `develop` branche qui contient les fonctionnalités planifiées.

Accès

Attribuez des autorisations selon le principe du moindre privilège. Les développeurs devraient avoir moins accès à l'environnement de test qu'à l'environnement de développement. Le personnel chargé de l'assurance qualité a besoin d'autorisations suffisantes pour tester la fonctionnalité.

Étapes de construction

Le processus de compilation dans cet environnement ne s'applique qu'aux corrections de bogues lors de l'utilisation de la stratégie Gitflow. La création d'une demande de fusion adressée à la `bugfix` branche lance automatiquement la construction.

1. Utilisez git-secrets (GitHub) pour rechercher des informations sensibles
2. Lint le code source
3. Compilez et compilez le code source, le cas échéant
4. Réaliser des tests unitaires
5. Réaliser une analyse de la couverture du code
6. Effectuez une analyse de code statique
7. Construisez iAc
8. Réaliser une analyse de sécurité IaC

9. Extraire des licences open source

Étapes de déploiement

Lancez automatiquement le déploiement de la `release` branche (Gitflow) ou de la `main` branche (Trunk ou GitHub Flow) dans l'environnement de test après le déploiement dans l'environnement de développement. Les étapes de déploiement dans l'environnement de test sont les suivantes :

1. Déployez la `release` branche (Gitflow) ou la `main` branche (Trunk ou GitHub Flow) dans l'environnement de test
2. Pause pour approbation manuelle par le personnel désigné
3. Télécharger les artefacts publiés
4. Effectuer le versionnement de la base de données
5. Effectuer le déploiement d'iAc
6. Réaliser des tests d'intégration
7. Réaliser des tests de performance
8. Approbation d'assurance qualité

Attentes avant de passer à l'environnement de mise en scène

- Les équipes de développement et d'assurance qualité ont effectué suffisamment de tests pour répondre aux exigences de votre organisation.
- L'équipe de développement a résolu tous les bogues découverts par le biais d'une `bugfix` branche.

Environnement de mise en scène

L'environnement de préparation est configuré pour être identique à l'environnement de production. Par exemple, l'étendue et la taille de la configuration des données doivent être similaires à celles des charges de travail de production. Utilisez l'environnement intermédiaire pour vérifier que le code et l'infrastructure fonctionnent comme prévu. Cet environnement est également le choix privilégié pour les cas d'utilisation professionnels, tels que les avant-premières ou les démonstrations destinées aux clients.

Accès

Attribuez des autorisations selon le principe du moindre privilège. Les développeurs doivent avoir le même accès à l'environnement de préparation qu'à l'environnement de production.

Étapes de construction

Aucune. Les mêmes artefacts que ceux utilisés dans l'environnement de test sont réutilisés dans l'environnement de préparation.

Étapes de déploiement

Lancez automatiquement le déploiement de la `release` branche (Gitflow) ou de la `main` branche (Trunk ou GitHub Flow) dans l'environnement de test après approbation et déploiement dans l'environnement de test. Les étapes de déploiement dans l'environnement de préparation sont les suivantes :

1. Déployez la `release` branche (Gitflow) ou `main` la branche (Trunk ou GitHub Flow) dans l'environnement intermédiaire
2. Pause pour approbation manuelle par le personnel désigné
3. Télécharger les artefacts publiés
4. Effectuer le versionnement de la base de données
5. Effectuer le déploiement d'iAc
6. (Facultatif) Effectuez des tests d'intégration
7. (Facultatif) Effectuer un test de charge
8. Obtenir l'approbation des approbateurs requis en matière de développement, d'assurance qualité, de produit ou d'entreprise

Attentes avant de passer à l'environnement de production

- Une version équivalente à la production a été déployée avec succès dans l'environnement de préparation
- (Facultatif) Les tests d'intégration et de charge ont été réussis

Environnement de production

L'environnement de production prend en charge le produit publié, en gérant les données réelles des clients réels. Il s'agit d'un environnement protégé auquel l'accès est attribué par le moindre privilège et l'accès élevé ne doit être autorisé que dans le cadre d'un processus d'exception audité pendant une période limitée.

Accès

Dans l'environnement de production, les développeurs doivent disposer d'un accès limité en lecture seule à l'AWS Management Console. Par exemple, les développeurs devraient être en mesure d'accéder aux données du journal pour les opérations quotidiennes. Toutes les mises en production doivent être soumises à une étape d'approbation avant le déploiement.

Étapes de construction

Aucune. Les mêmes artefacts utilisés dans les environnements de test et de préparation sont réutilisés dans l'environnement de production.

Étapes de déploiement

Lancez automatiquement le déploiement de la `release` branche (Gitflow) ou de la `main` branche (Trunk ou GitHub Flow) dans l'environnement de production après approbation et déploiement dans l'environnement de préparation. Les étapes de déploiement dans l'environnement de production sont les suivantes :

1. Déployez la `release` branche (Gitflow) ou la `main` branche (Trunk ou GitHub Flow) dans l'environnement de production
2. Pause pour approbation manuelle par le personnel désigné
3. Télécharger les artefacts publiés
4. Effectuer le versionnement de la base de données
5. Effectuer le déploiement d'iAc

Bonnes pratiques pour le développement basé sur Git

Pour réussir à adopter le développement basé sur Git, il est important de suivre un ensemble de bonnes pratiques qui favorisent la collaboration, maintiennent la qualité du code et soutiennent l'intégration et la livraison continues (CI/CD). Outre les meilleures pratiques décrites dans ce guide, consultez le guide AWS DevOps Well-Architected. Voici quelques bonnes pratiques clés pour le développement basé sur Git sur AWS :

- Limitez la fréquence et limitez les modifications : encouragez les développeurs à apporter des modifications ou des fonctionnalités mineures et incrémentielles. Cela réduit le risque de conflits de fusion et facilite l'identification et la résolution rapides des problèmes.
- Utiliser des boutons de fonctionnalité : pour gérer la publication de fonctionnalités incomplètes ou expérimentales, utilisez des boutons de fonctionnalité ou des indicateurs de fonctionnalité. Cela vous permet de masquer, d'activer ou de désactiver des fonctionnalités spécifiques en production sans affecter la stabilité de la branche principale.
- Maintenir une suite de tests robuste — Une suite de tests complète et bien entretenue est essentielle pour détecter les problèmes à un stade précoce et vérifier que la base de code reste stable. Investissez dans l'automatisation des tests et donnez la priorité à la correction des tests défaillants.
- Adoptez l'intégration continue : utilisez des outils et des pratiques d'intégration continue pour créer, tester et intégrer automatiquement les modifications de code dans la `develop` branche (Gitflow) ou la `main` branche (Trunk ou GitHub Flow). Cela vous permet de détecter les problèmes à un stade précoce et de rationaliser le processus de développement.
- Réaliser des révisions du code : encouragez les pairs à évaluer le code afin de maintenir la qualité, de partager les connaissances et de détecter les problèmes potentiels avant qu'ils ne soient intégrés à la `main` branche. Utilisez des pull requests ou d'autres outils de révision du code pour faciliter ce processus.
- Surveillez et corrigez les versions défectueuses : lorsqu'une version échoue ou que les tests échouent, donnez la priorité à la résolution du problème dès que possible. Cela permet de maintenir la `develop` branche (Gitflow) ou la `main` branche (Trunk ou GitHub Flow) dans un état libérable et de minimiser l'impact sur les autres développeurs.
- Communiquez et collaborez — Favorisez une communication ouverte et une collaboration entre les membres de l'équipe. Assurez-vous que les développeurs sont au courant des travaux en cours et des modifications apportées à la base de code.

- Refactorisation continue : refactorisez régulièrement la base de code pour améliorer sa maintenabilité et réduire la dette technique. Encouragez les développeurs à laisser le code dans un meilleur état que celui dans lequel ils l'ont trouvé.
- Utilisez des branches éphémères pour les tâches complexes : pour les tâches plus importantes ou plus complexes, utilisez des branches éphémères (également appelées branches de tâches) pour travailler sur les modifications. Cependant, veillez à ce que la durée de vie de la branche soit courte, généralement inférieure à une journée. Fusionnez les modifications dans la `develop` branche (Gitflow) ou dans la `main` branche (Trunk ou GitHub Flow) dès que possible. Les fusions et révisions plus petites et plus fréquentes sont plus faciles à utiliser et à traiter pour une équipe qu'une seule demande de fusion de grande envergure.
- Former et soutenir l'équipe — Fournir une formation et un soutien aux développeurs qui découvrent le développement basé sur Git ou qui ont besoin de conseils pour adopter ses meilleures pratiques.

Stratégies de branchement Git

De la plus simple à la plus complexe, ce guide décrit en détail les stratégies de Git-based branchement suivantes :

- Trunk : Trunk-based le développement est une pratique de développement logiciel dans laquelle tous les développeurs travaillent sur une seule branche, généralement appelée `main` branche `trunk` or. L'idée qui sous-tend cette approche est de maintenir la base de code dans un état continuellement publiable en intégrant fréquemment les modifications du code et en s'appuyant sur des tests automatisés et une intégration continue.
- GitHub Flow — GitHub Flow est un flux de travail léger basé sur les branches développé par. GitHub Il est basé sur l'idée de `feature` branches éphémères. Lorsqu'une fonctionnalité est terminée et prête à être déployée, elle est fusionnée dans la `main` branche.
- Gitflow — Avec une approche Gitflow, le développement est effectué dans des branches de fonctionnalités individuelles. Après approbation, vous fusionnez `feature` les branches dans une branche d'intégration généralement nommée`develop`. Lorsque suffisamment de fonctionnalités se sont accumulées dans la `develop` branche, une `release` branche est créée pour déployer les fonctionnalités dans les environnements supérieurs.

Chaque stratégie de branchement présente des avantages et des inconvénients. Bien qu'ils utilisent tous les mêmes environnements, ils n'utilisent pas tous les mêmes branches ni les mêmes étapes d'approbation manuelle. Dans cette section du guide, passez en revue chaque stratégie de succursale en détail afin de vous familiariser avec ses nuances et de déterminer si elle correspond au cas d'utilisation de votre organisation.

Rubriques de cette section :

- Stratégie de branchement des troncs
- GitHub Stratégie de branchement des flux
- Stratégie de branchement Gitflow

Stratégie de branchement des troncs

Trunk-based le développement est une pratique de développement logiciel dans laquelle tous les développeurs travaillent sur une seule branche, généralement appelée `main` branche `trunk` or.

L'idée qui sous-tend cette approche est de maintenir la base de code dans un état continuellement publiable en intégrant fréquemment les modifications du code et en s'appuyant sur des tests automatisés et une intégration continue.

Dans le développement basé sur les troncs, les développeurs valident leurs modifications dans la `main` branche plusieurs fois par jour, dans le but de procéder à de petites mises à jour incrémentielles. Cela permet des boucles de feedback rapides, réduit le risque de conflits de fusion et favorise la collaboration entre les membres de l'équipe. Cette pratique souligne l'importance d'une suite de tests bien entretenue, car elle repose sur des tests automatisés pour détecter rapidement les problèmes potentiels et garantir que la base de code reste stable et publiable.

Trunk-based le développement est souvent opposé au développement basé sur les fonctionnalités (également connu sous le nom de branche de fonctionnalités ou développement piloté par les fonctionnalités), où chaque nouvelle fonctionnalité ou correction de bogue est développée dans sa propre branche dédiée, distincte de la branche principale. Le choix entre le développement basé sur les troncs et le développement basé sur les fonctionnalités dépend de facteurs tels que la taille de l'équipe, les exigences du projet et l'équilibre souhaité entre la collaboration, la fréquence d'intégration et la gestion des versions.

Pour plus d'informations sur la stratégie de branchement Trunk, consultez les ressources suivantes :

- Mettre en œuvre une stratégie de branchement principal pour les DevOps environnements multi-comptes (directivesAWS prescriptives)
- Introduction au Trunk-Based développement (site Web de développement basé sur le tronc)

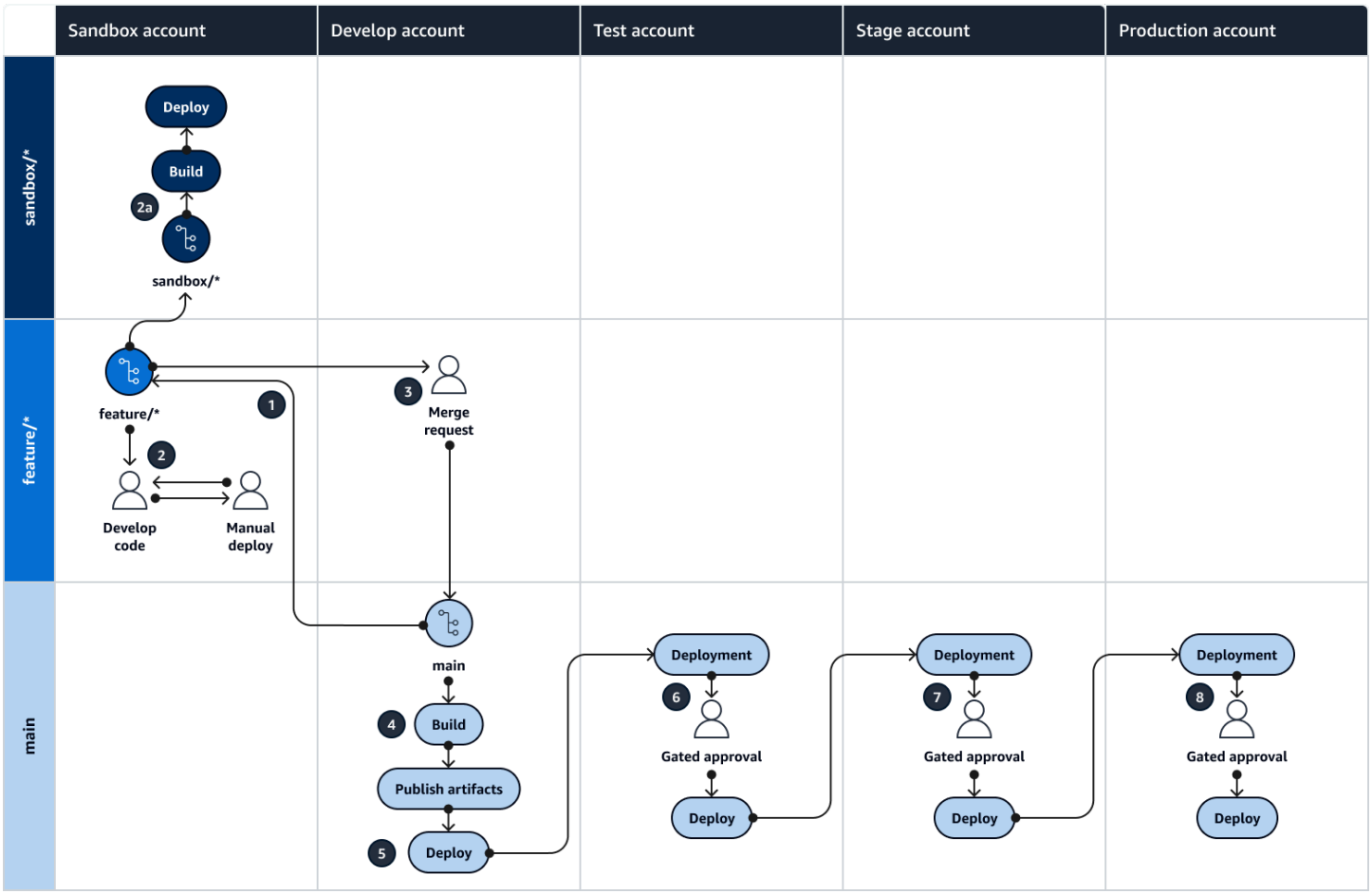
Rubriques de cette section :

- Aperçu visuel de la stratégie Trunk
- Stratégie Branches in a Trunk
- Avantages et inconvénients de la stratégie Trunk

Aperçu visuel de la stratégie Trunk

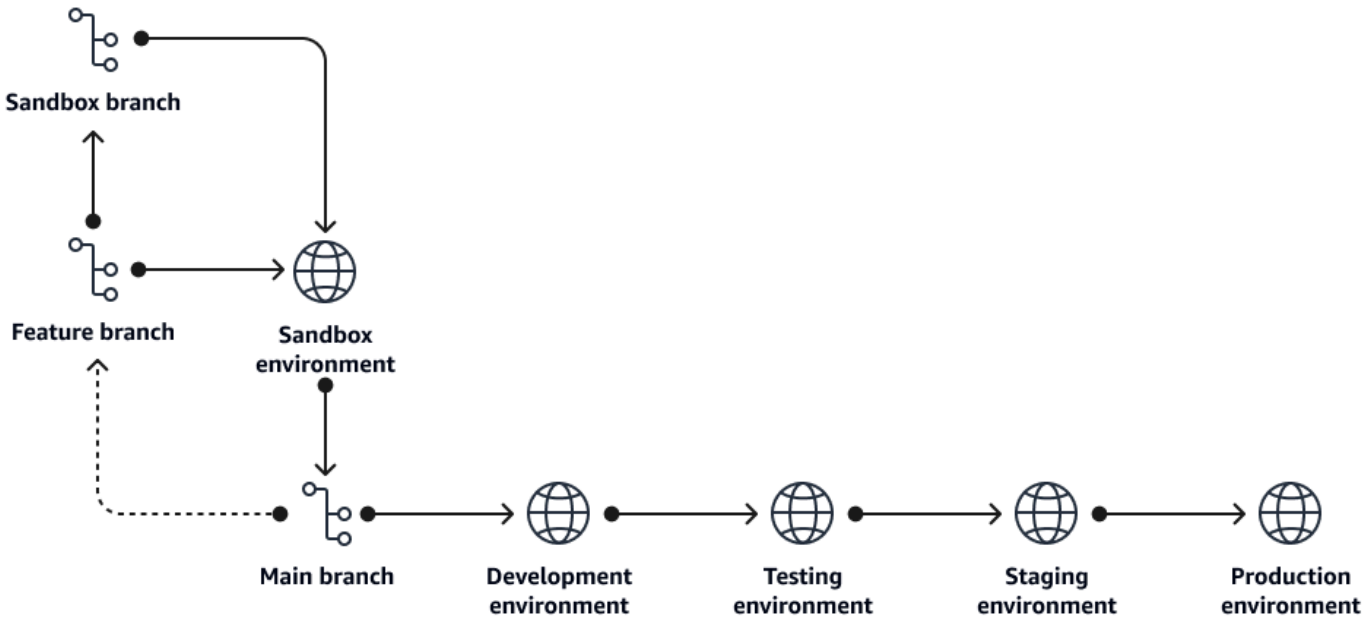
Le schéma suivant peut être utilisé comme un carré de Punnett (Wikipedia) pour comprendre la stratégie de branchement du tronc. Alignez les branches sur l'axe vertical avec les AWS environnements sur l'axe horizontal pour déterminer les actions à effectuer dans chaque scénario. Les chiffres encerclés vous guident dans la séquence d'actions représentée dans le diagramme. Ce

diagramme montre le flux de travail de développement d'une stratégie de branchement Trunk, depuis une `feature` branche dans l'environnement sandbox jusqu'à la version de production de la `main` branche. Pour plus d'informations sur les activités qui se produisent dans chaque environnement, consultez la section DevOps environnements de ce guide.



Stratégie Branches in a Trunk

Une stratégie de branchement Trunk comporte généralement les branches suivantes.



branche de fonctionnalités

Vous développez des fonctionnalités ou créez un correctif dans une `feature` branche. Pour créer une `feature` branche, vous devez dériver de la `main` branche. Les développeurs itèrent, valident et testent le code dans une `feature` branche. Lorsqu'une fonctionnalité est terminée, le développeur en fait la promotion. Il n'y a que deux voies à suivre à partir d'une `feature` branche :

- Fusionner dans la `sandbox` branche
- Créez une demande de fusion dans la `main` succursale

Convention de dénomination :	`feature/<story number>_<developer initials>_<descriptor>`
Exemple de convention de dénomination :	`feature/123456_MS_Implement_Feature_A`

branche de bac à sable

Cette branche est une branche principale non standard, mais elle est utile pour le développement de CI/CD pipelines. La `sandbox` branche est principalement utilisée aux fins suivantes :

- Effectuez un déploiement complet dans l'environnement sandbox à l'aide des pipelines CI/CD
- Développez et testez un pipeline avant de soumettre des demandes de fusion pour des tests complets dans un environnement inférieur, tel que le développement ou les tests.

`Sandbox`les succursales sont de nature temporaire et sont destinées à être éphémères. Ils doivent être supprimés une fois les tests spécifiques terminés.

Convention de dénomination :	`sandbox/<story number>_<developer initials>_<descriptor>`
Exemple de convention de dénomination :	`sandbox/123456_MS_Test_Pipeline_Deploy`

branche principale

La `main` branche représente toujours le code en cours d'exécution en production. Le code est dérivé`main`, développé, puis fusionné à `main` nouveau vers. Les déploiements depuis `main` peuvent cibler n'importe quel environnement. Pour éviter toute suppression, activez la protection de branche pour la `main` branche.

Convention de dénomination :	`main`

branche hotfix

Il n'existe aucune `hotfix` branche dédiée dans un flux de travail basé sur des troncs. Les correctifs utilisent des `feature` branches.

Avantages et inconvénients de la stratégie Trunk

La stratégie de branchement Trunk convient parfaitement aux équipes de développement plus petites, matures et dotées de solides compétences en communication. Cela fonctionne également bien si vous disposez de versions continues et continues des fonctionnalités de l'application. Il n'est pas adapté si vos équipes de développement sont nombreuses ou fragmentées ou si vous avez prévu de publier de nombreuses fonctionnalités. Des conflits de fusion se produiront dans ce modèle.

Sachez donc que la résolution des conflits de fusion est une compétence clé. Tous les membres de l'équipe doivent être formés en conséquence.

Avantages

Trunk-based le développement offre plusieurs avantages qui peuvent améliorer le processus de développement, rationaliser la collaboration et améliorer la qualité globale du logiciel. Voici quelques-uns des principaux avantages :

- Des boucles de feedback plus rapides : avec le développement basé sur les troncs, les développeurs intègrent fréquemment leurs modifications de code, souvent plusieurs fois par jour. Cela permet un feedback plus rapide concernant les problèmes potentiels et aide les développeurs à identifier et à résoudre les problèmes plus rapidement qu'ils ne le feraient dans un modèle de développement basé sur les fonctionnalités.
- Réduction des conflits de fusion : dans le développement basé sur les troncs, le risque de conflits de fusion importants et complexes est minimisé car les modifications sont intégrées en permanence. Cela permet de maintenir une base de code plus propre et de réduire le temps passé à résoudre les conflits. La résolution des conflits peut être à la fois chronophage et source d'erreurs dans le développement basé sur les fonctionnalités.
- Collaboration améliorée : Trunk-based le développement encourage les développeurs à travailler ensemble sur la même branche, ce qui favorise une meilleure communication et une meilleure collaboration au sein de l'équipe. Cela peut accélérer la résolution des problèmes et renforcer la cohésion de l'équipe.
- Révisions de code simplifiées — Les modifications de code étant moins importantes et plus fréquentes dans le développement basé sur des troncs, il peut être plus facile de mener des révisions de code approfondies. Les modifications mineures sont généralement plus faciles à comprendre et à examiner, ce qui permet d'identifier plus efficacement les problèmes potentiels et les améliorations.
- Intégration et livraison continues : le Trunk-based développement soutient les principes d'intégration continue et de livraison continue (CI/CD). En maintenant la base de code dans un état publiable et en intégrant fréquemment les modifications, les équipes peuvent plus facilement adopter des CI/CD pratiques, ce qui se traduit par des cycles de déploiement plus rapides et une amélioration de la qualité logicielle.
- Qualité de code améliorée — Grâce à des intégrations, des tests et des révisions de code fréquents, le développement basé sur les troncs peut contribuer à améliorer la qualité globale du

code. Les développeurs peuvent détecter et résoudre les problèmes plus rapidement, réduisant ainsi le risque d'accumulation de dettes techniques au fil du temps.

- Stratégie de succursales simplifiée : le Trunk-based développement simplifie la stratégie de succursales en réduisant le nombre de succursales à longue durée de vie. Cela peut faciliter la gestion et la maintenance de la base de code, en particulier pour les grands projets ou les équipes.

Inconvénients

Trunk-based le développement présente certains inconvénients, qui peuvent avoir un impact sur le processus de développement et la dynamique de l'équipe. Voici quelques inconvénients notables :

- Isolement limité — Comme tous les développeurs travaillent sur la même branche, leurs modifications sont immédiatement visibles par tous les membres de l'équipe. Cela peut entraîner des interférences ou des conflits, provoquer des effets secondaires imprévus ou endommager le bâtiment. En revanche, le développement basé sur les fonctionnalités isole mieux les modifications afin que les développeurs puissent travailler de manière plus indépendante.
- Pression accrue sur les tests : Trunk-based le développement repose sur une intégration continue et des tests automatisés pour détecter rapidement les problèmes. Cependant, cette approche peut exercer une forte pression sur l'infrastructure de test et nécessite une suite de tests bien entretenue. Si les tests ne sont pas complets ou fiables, cela peut entraîner des problèmes non détectés dans la branche principale.
- Moins de contrôle sur les versions : Trunk-based le développement vise à maintenir la base de code dans un état continuellement publiable. Bien que cela puisse être avantageux, il n'est pas toujours adapté aux projets dont le calendrier de publication est strict ou à ceux qui nécessitent la publication conjointe de fonctionnalités spécifiques. Feature-basedle développement permet de mieux contrôler quand et comment les fonctionnalités sont publiées.
- Taux de désabonnement de code — Les développeurs intégrant constamment les modifications dans la branche principale, le développement basé sur les troncs peut entraîner une augmentation du taux de désabonnement de code. Cela peut rendre difficile pour les développeurs de suivre l'état actuel de la base de code et peut être source de confusion lorsqu'ils tentent de comprendre l'effet des modifications récentes.
- Nécessite une solide culture d'équipe : Trunk-based le développement exige un haut niveau de discipline, de communication et de collaboration entre les membres de l'équipe. Cela peut être difficile à maintenir, en particulier dans les grandes équipes ou lorsque vous travaillez avec des développeurs moins expérimentés avec cette approche.

- Défis d'évolutivité — À mesure que la taille de l'équipe de développement augmente, le nombre de modifications de code intégrées dans la branche principale peut augmenter rapidement. Cela peut entraîner des interruptions de compilation et des échecs de test plus fréquents, ce qui rend difficile le maintien de la base de code dans un état libérable.

GitHub Stratégie de branchement des flux

GitHub Flow est un flux de travail léger basé sur les branches développé par. GitHub GitHub Flow est basé sur l'idée de branches d'entités de courte durée qui sont fusionnées dans la branche principale lorsque la fonctionnalité est terminée et prête à être déployée. Les principes clés de GitHub Flow sont les suivants :

- Le branchement est léger : les développeurs peuvent créer des branches de fonctionnalités pour leur travail en quelques clics, ce qui améliore la capacité de collaboration et d'expérimentation sans affecter la branche principale.
- Déploiement continu — Les modifications sont déployées dès qu'elles sont fusionnées dans la branche principale, ce qui permet un retour d'information et une itération rapides.
- Demandes de fusion : les développeurs utilisent des demandes de fusion pour lancer un processus de discussion et de révision avant de fusionner leurs modifications dans la branche principale.

Pour plus d'informations sur GitHub Flow, consultez les ressources suivantes :

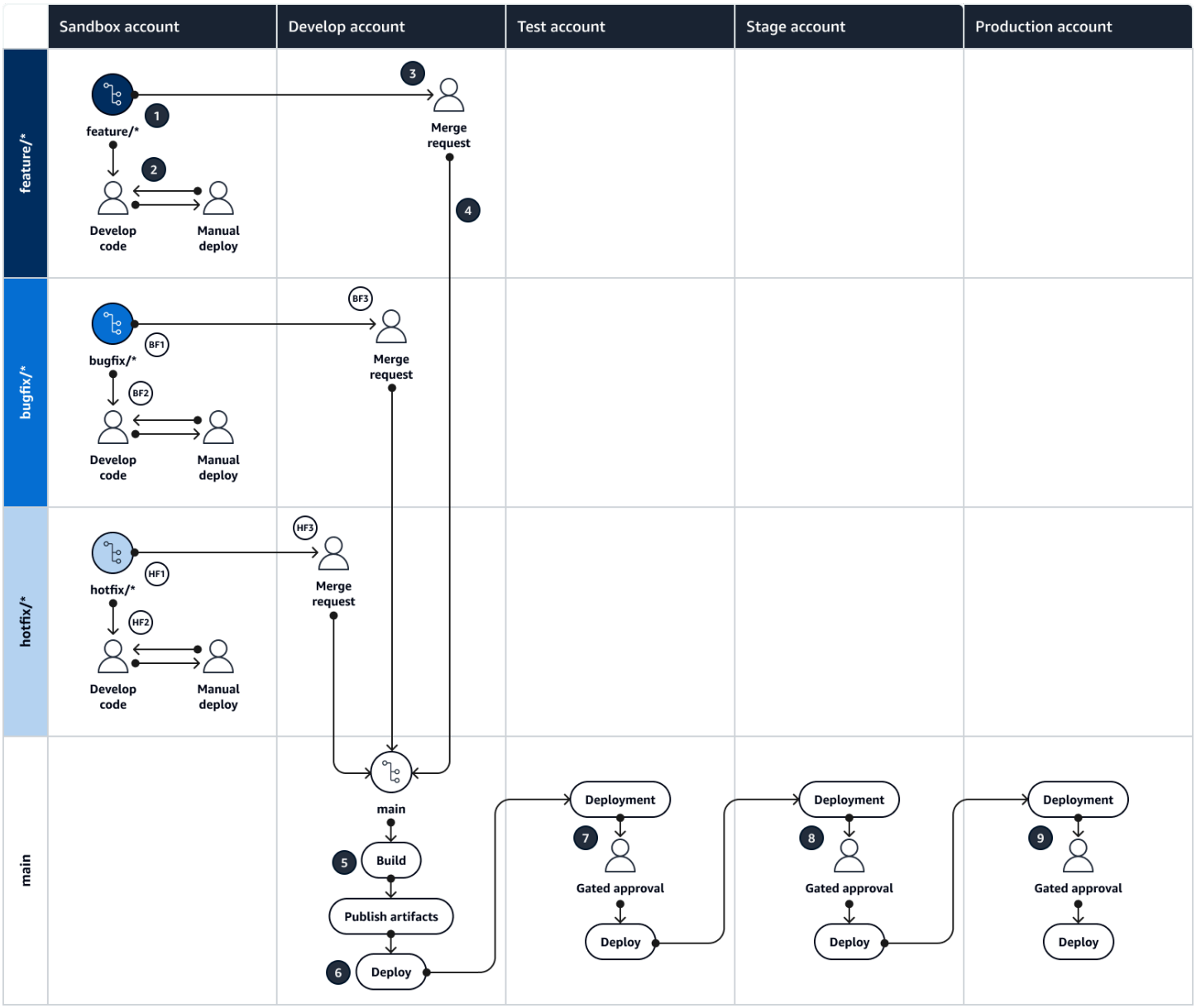
- Mettre en œuvre une stratégie de branchement GitHub Flow pour les DevOps environnements multi-comptes (directivesAWS prescriptives)
- GitHub Démarrage rapide de Flow (GitHub documentation)
- Pourquoi GitHub Flow ? (Site Web GitHub de Flow)

Rubriques de cette section :

- Aperçu visuel de la stratégie GitHub Flow
- Branches dans une stratégie GitHub Flow
- Avantages et inconvénients de la stratégie GitHub Flow

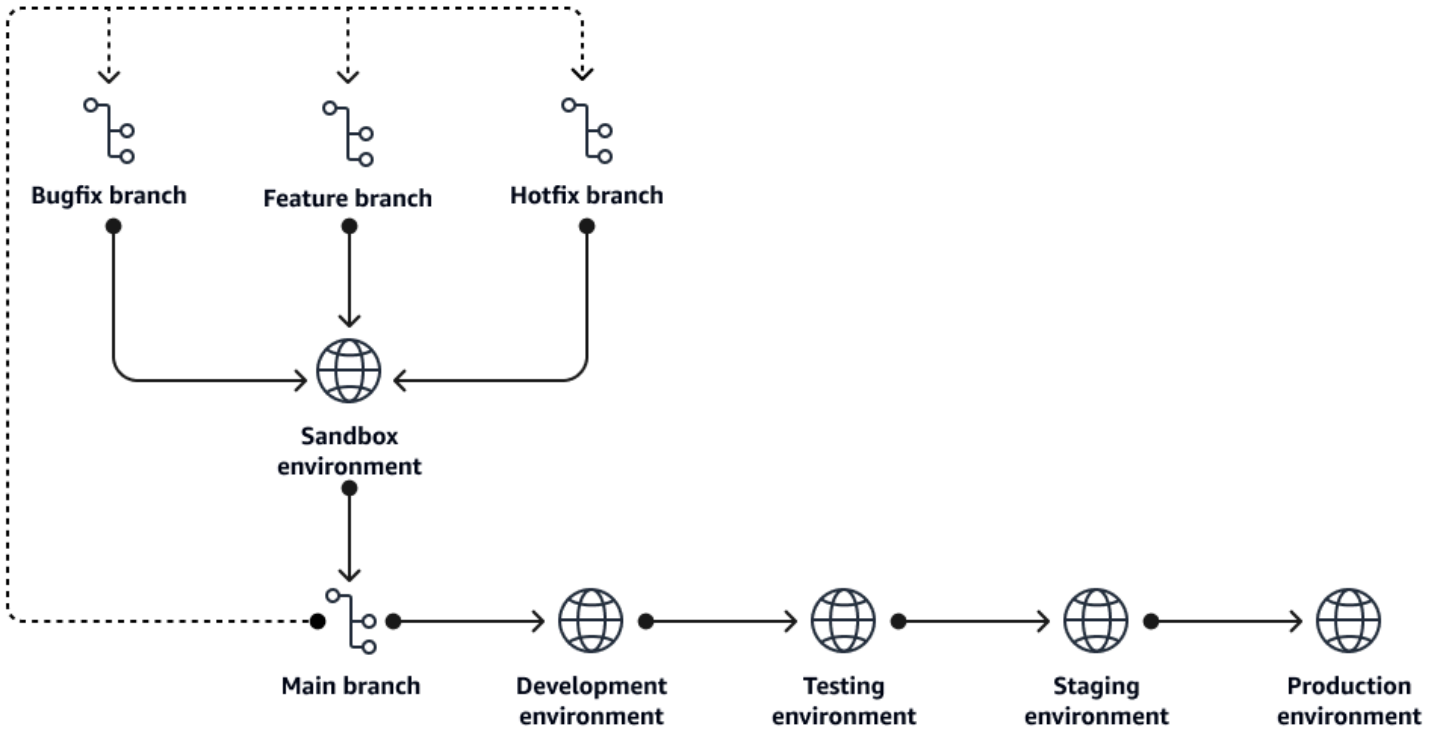
Aperçu visuel de la stratégie GitHub Flow

Le schéma suivant peut être utilisé comme un carré de Punnett pour comprendre la stratégie de branchement de GitHub Flow. Alignez les branches sur l'axe vertical avec les AWS environnements sur l'axe horizontal pour déterminer les actions à effectuer dans chaque scénario. Les chiffres encerclés vous guident dans la séquence d'actions représentée dans le diagramme. Ce diagramme montre le flux de travail de développement d'une stratégie de branchement GitHub Flow, depuis une branche fonctionnelle dans l'environnement sandbox jusqu'à la version de production de la branche principale. Pour plus d'informations sur les activités qui se produisent dans chaque environnement, consultez la section DevOps environnements de ce guide.



Branches dans une stratégie GitHub Flow

Une stratégie de branchement GitHub Flow comporte généralement les branches suivantes.



branche de fonctionnalités

Vous développez des fonctionnalités dans `feature` les succursales. Pour créer une `feature` branche, vous devez dériver de la `main` branche. Les développeurs itèrent, valident et testent le code dans la `feature` branche. Lorsqu'une fonctionnalité est terminée, le développeur en fait la promotion en créant une demande de fusion auprès de`main`.

Convention de dénomination :	`feature/<story number>_<developer initials>_<descriptor>`
Exemple de convention de dénomination :	`feature/123456_MS_Implement_Feature_A`

branche bugfix

La `bugfix` branche est utilisée pour résoudre les problèmes. Ces branches sont dérivées de la `main` branche. Une fois que le correctif a été testé dans le bac à sable ou dans l'un des environnements inférieurs, il peut être promu vers des environnements supérieurs en le fusionnant `main` via une demande de fusion. Il s'agit d'une convention de dénomination suggérée pour l'organisation et le suivi. Ce processus peut également être géré à l'aide d'une branche de fonctionnalités.

Convention de dénomination :	`bugfix/<ticket number>_<developer initials>_<descriptor>`
Exemple de convention de dénomination :	`bugfix/123456_MS_Fix_Problem_A`

branche hotfix

La `hotfix` branche est utilisée pour résoudre les problèmes critiques à fort impact avec un délai minimal entre le personnel de développement et le déploiement du code en production. Ces branches sont dérivées de la `main` branche. Une fois le correctif testé dans le bac à sable ou dans l'un des environnements inférieurs, il peut être promu vers des environnements supérieurs en le fusionnant `main` via une demande de fusion. Il s'agit d'une convention de dénomination suggérée pour l'organisation et le suivi. Ce processus peut également être géré à l'aide d'une branche de fonctionnalités.

Convention de dénomination :	`hotfix/<ticket number>_<developer initials>_<descriptor>`
Exemple de convention de dénomination :	`hotfix/123456_MS_Fix_Problem_A`

branche principale

La `main` branche représente toujours le code en cours d'exécution en production. Le code est fusionné dans la `main` branche à partir des `feature` branches à l'aide de demandes de fusion. Pour vous protéger contre la suppression et empêcher les développeurs d'y envoyer du code directement`main`, activez la protection de la `main` branche.

Convention de dénomination :	`main`

Avantages et inconvénients de la stratégie GitHub Flow

La stratégie de branchement de Github Flow convient parfaitement aux petites équipes de développement matures dotées de solides compétences en communication. Cette stratégie convient parfaitement aux équipes qui souhaitent mettre en œuvre une livraison continue, et elle est bien prise en charge par des CI/CD moteurs courants. GitHub Flow est léger, ne comporte pas trop de règles et est capable de soutenir des équipes qui évoluent rapidement. Il n'est pas adapté si vos équipes doivent suivre des processus de conformité ou de publication stricts. Les conflits de fusion sont courants dans ce modèle et se produiront probablement souvent. La résolution des conflits de fusion est une compétence essentielle, et vous devez former tous les membres de l'équipe en conséquence.

Avantages

GitHub Flow offre plusieurs avantages qui peuvent améliorer le processus de développement, rationaliser la collaboration et améliorer la qualité globale du logiciel. Voici quelques-uns des principaux avantages :

- Flexible et léger — GitHub Flow est un flux de travail léger et flexible qui aide les développeurs à collaborer sur des projets de développement de logiciels. Il permet une itération et une expérimentation rapides avec un minimum de complexité.
- Collaboration simplifiée — GitHub Flow fournit un processus clair et rationalisé pour gérer le développement des fonctionnalités. Il encourage de petits changements ciblés qui peuvent être rapidement revus et fusionnés, améliorant ainsi l'efficacité.
- Contrôle de version clair — Avec GitHub Flow, chaque modification est effectuée dans une branche distincte. Cela permet d'établir un historique de contrôle de version clair et traçable. Cela permet aux développeurs de suivre et de comprendre les modifications, de revenir en arrière si nécessaire et de maintenir une base de code fiable.
- Intégration continue fluide — GitHub Flow s'intègre aux outils d'intégration continue. La création de pull requests peut lancer des processus de test et de déploiement automatisés. Les outils CI vous aident à tester de manière approfondie les modifications avant qu'elles ne soient fusionnées dans la `main` branche, réduisant ainsi le risque d'introduction de bogues dans la base de code.

- Feedback rapide et amélioration continue — GitHub Flow encourage une boucle de feedback rapide en promouvant des révisions de code fréquentes et des discussions par le biais de pull requests. Cela facilite la détection précoce des problèmes, favorise le partage des connaissances entre les membres de l'équipe et conduit finalement à une meilleure qualité du code et à une meilleure collaboration au sein de l'équipe de développement.
- Annulations et annulations simplifiées : dans le cas où une modification de code introduirait un bogue ou un problème inattendu, GitHub Flow simplifie le processus d'annulation ou d'annulation de la modification. En disposant d'un historique clair des validations et des branches, il est plus facile d'identifier et d'annuler les modifications problématiques, ce qui contribue à maintenir une base de code stable et fonctionnelle.
- Courbe d'apprentissage légère — GitHub Flow peut être plus facile à apprendre et à adopter que Gitflow, en particulier pour les équipes déjà familiarisées avec Git et les concepts de contrôle de version. Sa simplicité et son modèle de branchement intuitif le rendent accessible aux développeurs de différents niveaux d'expérience, réduisant ainsi la courbe d'apprentissage associée à l'adoption de nouveaux flux de travail de développement.
- Développement continu — GitHub Flow permet aux équipes d'adopter une approche de déploiement continu en permettant le déploiement immédiat de chaque modification dès son intégration dans la `main` succursale. Ce processus rationalisé élimine les retards inutiles et garantit que les dernières mises à jour et améliorations sont rapidement mises à la disposition des utilisateurs. Cela se traduit par un cycle de développement plus agile et plus réactif.

Inconvénients

Bien que GitHub Flow présente plusieurs avantages, il est important de prendre également en compte ses inconvénients potentiels :

- Adaptation limitée aux grands projets — GitHub Flow peut ne pas être aussi adapté aux projets de grande envergure comportant des bases de code complexes et de multiples branches de fonctionnalités à long terme. Dans de tels cas, un flux de travail plus structuré, tel que Gitflow, peut permettre de mieux contrôler le développement simultané et la gestion des versions.
- Absence de structure de version officielle — GitHub Flow ne définit pas explicitement un processus de publication ou ne prend pas en charge les fonctionnalités telles que le contrôle de version, les correctifs ou les branches de maintenance. Cela peut constituer une limite pour les projets nécessitant une gestion stricte des versions ou nécessitant un support et une maintenance à long terme.

- Support limité pour la planification des versions à long terme : GitHub Flow se concentre sur les branches de fonctionnalités éphémères, qui peuvent ne pas convenir aux projets nécessitant une planification des versions à long terme, tels que ceux dotés de feuilles de route strictes ou de dépendances étendues entre les fonctionnalités. La gestion de calendriers de publication complexes peut s'avérer difficile compte tenu des contraintes de GitHub Flow.
- Risque de conflits de fusion fréquents — Comme GitHub Flow encourage les branchements et les fusions fréquents, il est possible de rencontrer des conflits de fusion, en particulier dans les projets impliquant une forte activité de développement. La résolution de ces conflits peut prendre beaucoup de temps et nécessiter des efforts supplémentaires de la part de l'équipe de développement.
- Absence de phases de flux de travail formalisées — GitHub Flow ne définit pas de phases de développement explicites, telles que les étapes alpha, bêta ou de version candidate. Cela peut rendre plus difficile la communication de l'état actuel du projet ou du niveau de stabilité des différentes branches ou versions.
- Impact des modifications radicales : étant donné que GitHub Flow encourage la fusion fréquente des modifications dans la `main` branche, le risque d'introduire des modifications majeures affectant la stabilité de la base de code est accru. Des pratiques strictes de révision du code et de test sont essentielles pour atténuer efficacement ce risque.

Stratégie de branchement Gitflow

Gitflow est un modèle de branchement qui implique l'utilisation de plusieurs branches pour faire passer le code du développement à la production. Gitflow fonctionne bien pour les équipes qui ont des cycles de publication planifiés et qui ont besoin de définir un ensemble de fonctionnalités en tant que version. Le développement est effectué dans des branches de fonctionnalités individuelles qui sont fusionnées, avec approbation, dans une branche de développement, qui est utilisée pour l'intégration. Les fonctionnalités de cette branche sont considérées comme prêtes à être produites. Lorsque toutes les fonctionnalités planifiées sont accumulées dans la branche de développement, une branche de publication est créée pour les déploiements dans des environnements supérieurs. Cette séparation permet de mieux contrôler quelles modifications sont transférées vers tel ou tel environnement nommé selon un calendrier défini. Si nécessaire, vous pouvez accélérer ce processus pour obtenir un modèle de déploiement plus rapide.

Pour plus d'informations sur la stratégie de branchement de Gitflow, consultez les ressources suivantes :

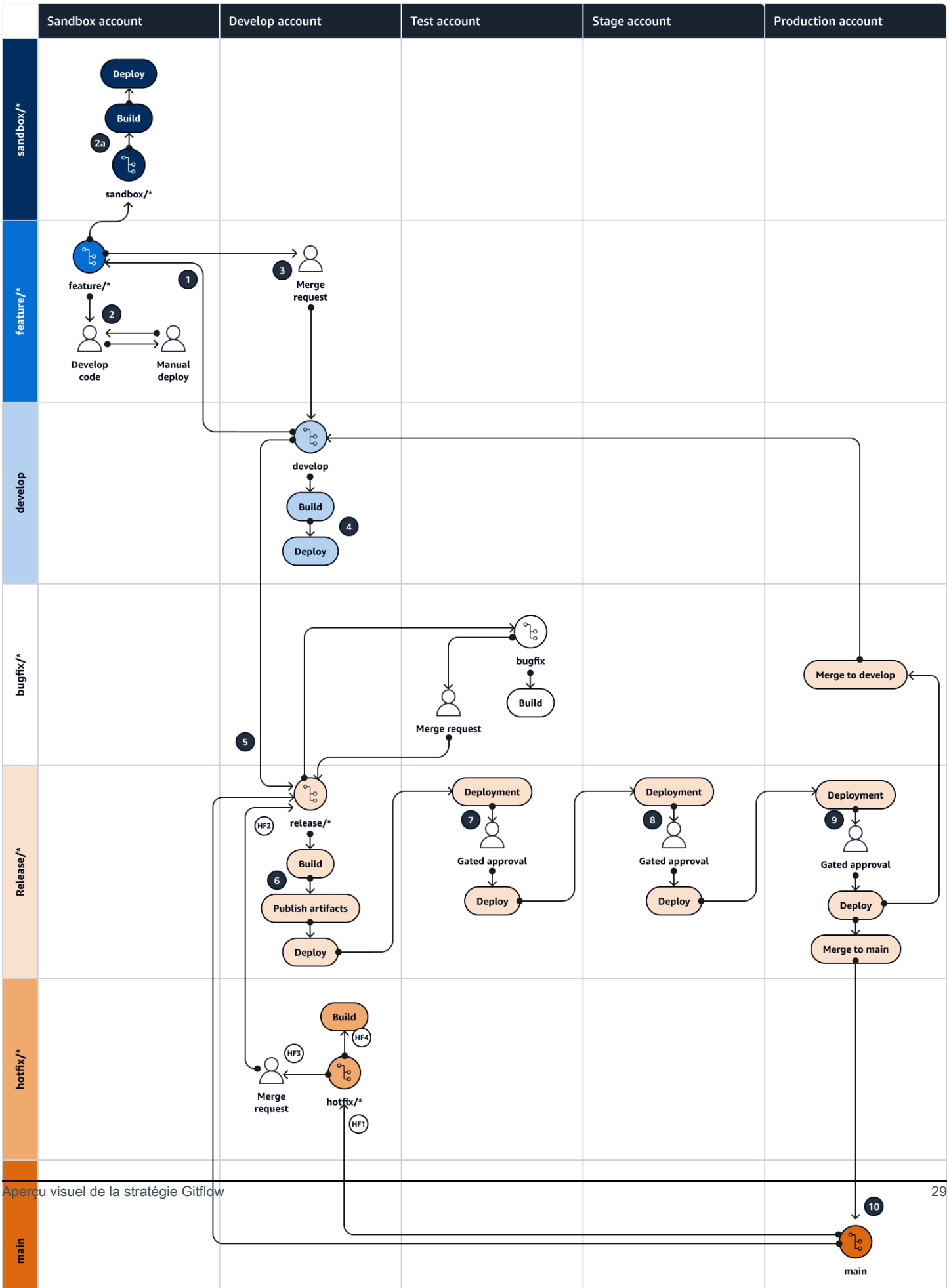
- Mettre en œuvre une stratégie de branchement Gitflow pour les DevOps environnements multi-comptes (directives prescriptives)AWS
- Le blog original de Gitflow (article de blog de Vincent Driessen)
- Flux de travail Gitflow (Atlassian)

Rubriques de cette section :

- Aperçu visuel de la stratégie Gitflow
- Branches dans une stratégie Gitflow
- Avantages et inconvénients de la stratégie Gitflow

Aperçu visuel de la stratégie Gitflow

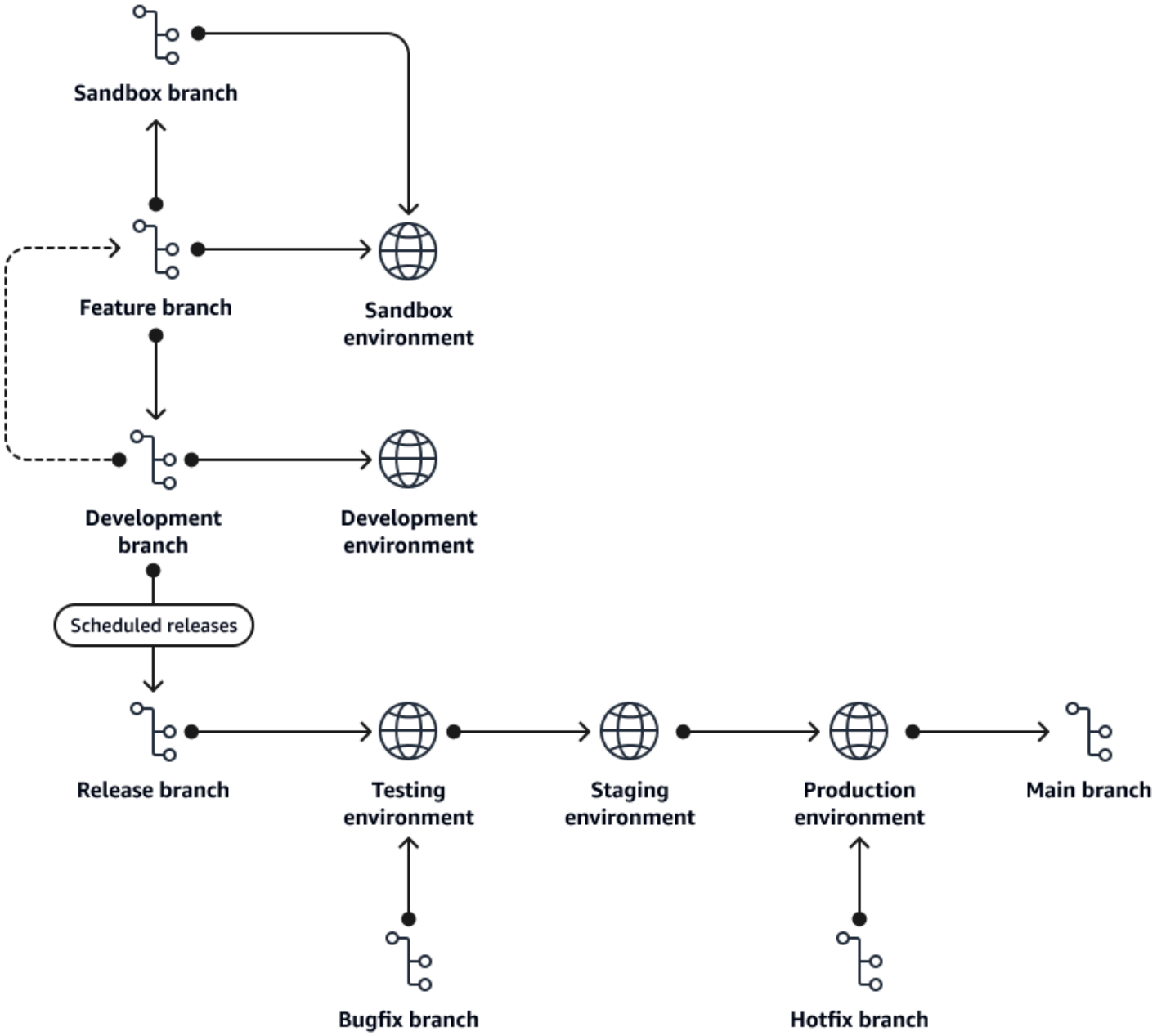
Le schéma suivant peut être utilisé comme un carré de Punnett pour comprendre la stratégie de branchement de Gitflow. Alignez les branches sur l'axe vertical avec les AWS environnements sur l'axe horizontal pour déterminer les actions à effectuer dans chaque scénario. Les chiffres encerclés vous guident dans la séquence d'actions représentée dans le diagramme. Pour plus d'informations sur les activités qui se produisent dans chaque environnement, consultez la section DevOps environnements de ce guide.



Aperçu visuel de la stratégie Gitflow

Branches dans une stratégie Gitflow

Une stratégie de branchement Gitflow comporte généralement les branches suivantes.



branche de fonctionnalités

`Feature`les branches sont des branches à court terme dans lesquelles vous développez des fonctionnalités. La `feature` branche est créée en partant de la `develop` branche. Les développeurs itèrent, valident et testent le code dans la `feature` branche. Lorsque la fonctionnalité est terminée,

le développeur en fait la promotion. Il n'existe que deux voies de transfert à partir d'une branche de fonctionnalités :

- Fusionner dans la `sandbox` branche
- Créez une demande de fusion dans la `develop` succursale

Convention de dénomination :	`feature/<story number>_<developer initials>_<descriptor>`
Exemple de convention de dénomination :	`feature/123456_MS_Implement_Feature_A`

branche de bac à sable

La `sandbox` branche est une branche non standard à court terme pour Gitflow. Cependant, il est utile pour le développement de CI/CD pipelines. La `sandbox` branche est principalement utilisée aux fins suivantes :

- Effectuez un déploiement complet dans l'environnement sandbox en utilisant les CI/CD pipelines plutôt qu'un déploiement manuel.
- Développez et testez un pipeline avant de soumettre des demandes de fusion pour des tests complets dans un environnement inférieur, tel que le développement ou les tests.

`Sandbox`les branches sont de nature temporaire et ne sont pas destinées à durer longtemps. Ils doivent être supprimés une fois les tests spécifiques terminés.

Convention de dénomination :	`sandbox/<story number>_<developer initials>_<descriptor>`
Exemple de convention de dénomination :	`sandbox/123456_MS_Test_Pipeline_Deploy`

développer une branche

La `develop` branche est une branche durable dans laquelle les fonctionnalités sont intégrées, créées, validées et déployées dans l'environnement de développement. Toutes les `feature` branches sont fusionnées dans la `develop` branche. Les fusions avec la `develop` branche sont effectuées par le biais d'une demande de fusion qui nécessite une compilation réussie et deux approbations de développeurs. Pour empêcher la suppression, activez la protection des branches sur la `develop` branche.

Convention de dénomination :	`develop`

branche de sortie

Dans Gitflow, `release` les branches sont des branches à court terme. Ces branches sont spéciales car vous pouvez les déployer dans plusieurs environnements, en adoptant la méthodologie Build-Once, Deploy-Many. `Release`les branches peuvent cibler les environnements de test, de préparation ou de production. Une fois qu'une équipe de développement a décidé de promouvoir des fonctionnalités dans des environnements supérieurs, elle crée une nouvelle `release` branche et utilise le numéro de version incrémenté par rapport à la version précédente. Au niveau de chaque environnement, les déploiements nécessitent des approbations manuelles pour pouvoir être effectués. `Release`les branches doivent nécessiter la modification d'une demande de fusion.

Une fois que la `release` branche a été déployée en production, elle doit être fusionnée à nouveau dans les `main` branches `develop` et pour s'assurer que les corrections de bogues ou les correctifs seront intégrés dans les futurs efforts de développement.

Convention de dénomination :	`release/v{major}.{minor}`
Exemple de convention de dénomination :	`release/v1.0`

branche principale

La `main` branche est une branche à longue durée de vie qui représente toujours le code en cours d'exécution en production. Le code est automatiquement fusionné dans la `main` branche à partir d'une branche de version après un déploiement réussi depuis le pipeline de publication. Pour empêcher la suppression, activez la protection des branches sur la `main` branche.

Convention de dénomination :	`main`

branche bugfix

La `bugfix` branche est une branche à court terme qui est utilisée pour résoudre les problèmes dans les branches de publication qui n'ont pas été mises en production. Une `bugfix` branche ne doit être utilisée que pour promouvoir les corrections apportées dans `release` les branches aux environnements de test, de test ou de production. Une `bugfix` branche est toujours dérivée d'une `release` branche.

Une fois le correctif testé, il peut être promu vers la `release` branche par le biais d'une demande de fusion. Vous pouvez ensuite faire avancer la `release` branche en suivant le processus de publication standard.

Convention de dénomination :	`bugfix/<ticket>_<developer initials>_<descriptor>`
Exemple de convention de dénomination :	`bugfix/123456_MS_Fix_Problem_A`

branche hotfix

La `hotfix` succursale est une succursale à court terme utilisée pour résoudre les problèmes de production. Il ne doit être utilisé que pour promouvoir des correctifs qui doivent être accélérés pour atteindre l'environnement de production. Une `hotfix` branche est toujours issue d'une branche. `main`

Une fois le correctif testé, vous pouvez le promouvoir en production par le biais d'une demande de fusion dans la `release` branche à partir `main` de laquelle il a été créé. Pour les tests, vous pouvez ensuite faire avancer la `release` branche en suivant le processus de publication standard.

Convention de dénomination :	`hotfix/<ticket>_<developer initials>_<descriptor>`
Exemple de convention de dénomination :	`hotfix/123456_MS_Fix_Problem_A`

Avantages et inconvénients de la stratégie Gitflow

La stratégie de branchement de Gitflow convient parfaitement aux équipes plus importantes et plus distribuées qui ont des exigences strictes en matière de publication et de conformité. Gitflow contribue à un cycle de publication prévisible pour l'organisation, ce qui est souvent préféré par les grandes entreprises. Gitflow convient également aux équipes qui ont besoin de garde-fous pour terminer correctement leur cycle de vie de développement logiciel. Cela s'explique par le fait que la stratégie comporte de nombreuses opportunités d'évaluation et d'assurance qualité. Gitflow convient également aux équipes qui doivent gérer simultanément plusieurs versions de versions de production. Certains inconvénients de GitFlow sont qu'il est plus complexe que les autres modèles de branchement et qu'il nécessite un strict respect du modèle pour réussir. Gitflow ne fonctionne pas bien pour les organisations qui recherchent une livraison continue en raison de la nature rigide de la gestion des branches de publication. Les agences de lancement de Gitflow peuvent être des succursales à longue durée de vie susceptibles d'accumuler des dettes techniques si elles ne sont pas traitées correctement en temps opportun.

Avantages

Gitflow-based le développement offre plusieurs avantages qui peuvent améliorer le processus de développement, rationaliser la collaboration et améliorer la qualité globale du logiciel. Voici quelques-uns des principaux avantages :

- Processus de publication prévisible — Gitflow suit un processus de publication régulier et prévisible. Il convient parfaitement aux équipes ayant des cadences de développement et de publication régulières.
- Collaboration améliorée — Gitflow encourage l'utilisation de `feature` et de `release` branches. Ces deux branches permettent aux équipes de travailler en parallèle avec un minimum de dépendance les unes par rapport aux autres.
- Bien adapté à plusieurs environnements, Gitflow utilise des `release` branches, qui peuvent avoir une durée de vie plus longue. Ces branches permettent aux équipes de cibler des versions individuelles sur une plus longue période.
- Plusieurs versions en production — Si votre équipe prend en charge plusieurs versions du logiciel en production, les `release` succursales de Gitflow prennent en charge cette exigence.
- Built-in évaluations de la qualité du code — Gitflow exige et encourage l'utilisation de révisions et d'approbations de code avant que le code ne soit promu dans un autre environnement. Ce

processus élimine les frictions entre les développeurs en imposant cette étape pour toutes les promotions de code.

- Avantages pour l'organisation — Gitflow présente également des avantages au niveau de l'organisation. Gitflow encourage l'utilisation d'un cycle de publication standard, qui aide l'organisation à comprendre et à anticiper le calendrier de publication. Comme l'entreprise sait désormais quand de nouvelles fonctionnalités peuvent être proposées, les délais sont réduits grâce à des dates de livraison fixes.

Inconvénients

Gitflow-based le développement présente certains inconvénients qui peuvent avoir un impact sur le processus de développement et la dynamique de l'équipe. Voici quelques inconvénients notables :

- Complexité — Gitflow est un modèle complexe que les nouvelles équipes doivent apprendre, et vous devez respecter les règles de Gitflow pour l'utiliser avec succès.
- Déploiement continu — Gitflow ne convient pas à un modèle dans lequel de nombreux déploiements sont rapidement mis en production. Cela est dû au fait que Gitflow nécessite l'utilisation de plusieurs branches et un flux de travail strict régissant la `release` branche.
- Gestion des succursales — Gitflow utilise de nombreuses branches, dont la maintenance peut s'avérer fastidieuse. Il peut être difficile de suivre les différentes branches et de fusionner le code publié afin de maintenir les branches correctement alignées les unes avec les autres.
- Dette technique — Les versions de Gitflow étant généralement plus lentes que les autres modèles de branchement, un plus grand nombre de fonctionnalités peuvent s'accumuler au fur et à mesure de leur publication, ce qui peut entraîner une accumulation de dettes techniques.

Les équipes doivent soigneusement tenir compte de ces inconvénients lorsqu'elles décident si le Gitflow-based développement est la bonne approche pour leur projet.

Étapes suivantes

Ce guide explique les différences entre trois stratégies de branchement Git courantes : GitHub Flow, Gitflow et Trunk. Il décrit leurs flux de travail en détail et présente également les avantages et les inconvénients de chacun. Les étapes suivantes consistent à choisir l'un de ces flux de travail standard pour votre organisation. Pour mettre en œuvre l'une de ces stratégies de branchement, consultez les rubriques suivantes :

- Mettre en œuvre une stratégie de branchement Trunk pour les environnements multi-comptes DevOps
- Mettre en œuvre une stratégie GitHub de branchement Flow pour les environnements multi-comptes DevOps
- Mettre en œuvre une stratégie de branchement Gitflow pour les environnements multi-comptes DevOps

Si vous ne savez pas par où commencer l'utilisation de Git et des DevOps processus par votre équipe, nous vous recommandons de choisir une solution standard et de la tester. L'utilisation d'une convention de branchement standard permet à l'équipe de s'appuyer sur la documentation existante et de découvrir ce qui fonctionne le mieux pour elle.

N'ayez pas peur de modifier votre stratégie si elle ne fonctionne pas pour votre organisation ou vos équipes de développement. Les besoins et les exigences des équipes de développement peuvent évoluer au fil du temps, et il n'existe pas de solution unique et parfaite.

Ressources

Ce guide n'inclut pas de formation à Git ; toutefois, de nombreuses ressources de haute qualité sont disponibles sur Internet si vous avez besoin de cette formation. Nous vous recommandons de commencer par le site de documentation Git.

Les ressources suivantes peuvent vous aider dans votre parcours de branchement Git dans le AWS Cloud.

AWS Conseils prescriptifs

- Mettre en œuvre une stratégie de branchement Trunk pour les environnements multi-comptes DevOps
- Mettre en œuvre une stratégie GitHub de branchement Flow pour les environnements multi-comptes DevOps
- Mettre en œuvre une stratégie de branchement Gitflow pour les environnements multi-comptes DevOps

Autres AWS conseils

- AWS DevOps Orientations
- AWS Architecture de référence du pipeline de déploiement
- Présentation de DevOps
- DevOpsressources

Autres ressources

- Méthodologie d'application à douze facteurs (12factor.net)
- Git-Secrets (1) GitHub
- Gitflow
  - Le blog original de Gitflow (article de blog de Vincent Driessen)
  - Flux de travail Gitflow (Atlassian)

  - Gitflow on GitHub : Comment utiliser les flux de travail Git Flow avec GitHub Based Repos (YouTubevidéo)
  - Exemple d'initialisation de Git Flow (YouTube vidéo)
  - La branche de lancement de Gitflow du début à la fin (YouTube vidéo)
- GitHub Flux
  - GitHub Démarrage rapide de Flow (GitHub documentation)
  - Pourquoi GitHub Flow ? (Site Web GitHub de Flow)
- Tronc
  - Introduction au développement basé sur les troncs (site Web de développement basé sur les troncs)

Collaborateurs

Conception

- Mike Stephens, architecte senior des applications cloud, AWS
- Rayjan Wilson, architecte senior d'applications cloud, AWS
- Abhilash Vinod, chef d'équipe, architecte senior d'applications cloud, AWS

Révision

- Stephen DiCato, consultant principal en sécurité, AWS
- Gaurav Samudra, architecte d'applications cloud, AWS
- Steven Guggenheimer, chef d'équipe, architecte senior d'applications cloud, AWS

Rédaction technique

- Lilly AbouHarb, rédactrice technique senior, AWS

Historique du document

Le tableau suivant décrit les modifications importantes apportées à ce guide. Pour être averti des mises à jour à venir, abonnez-vous à un fil RSS.

Modification	Description	Date
Mettre à jour	Nous avons remplacé les images dans les sections Branches dans une stratégie Trunk, Branches dans une stratégie GitHub Flow et Branches dans une stratégie Gitflow.	5 juin 2026
Publication initiale	—	15 février 2024

Les traductions sont fournies par des outils de traduction automatique. En cas de conflit entre le contenu d'une traduction et celui de la version originale en anglais, la version anglaise prévaudra.