Auswahl einer Git-Branching-Strategie für Umgebungen mit mehreren Konten DevOps

AWS Prescriptive Guidance

AWS Prescriptive Guidance: Auswahl einer Git-Branching-Strategie für Umgebungen mit mehreren Konten DevOps

Table of Contents

Auswahl einer Git-Branching-Strategie für Umgebungen mit mehreren Konten DevOps

Amazon Web Services (Mitwirkende)

Februar 2024 (Verlauf der Dokumente)

Die Umstellung auf einen cloudbasierten Ansatz und die Bereitstellung von Softwarelösungen AWS kann transformativ sein. Möglicherweise sind Änderungen an Ihrem Lebenszyklusprozess für die Softwareentwicklung erforderlich. In der Regel AWS-Konten werden während des Entwicklungsprozesses mehrere verwendet AWS Cloud. Die Wahl einer kompatiblen Git-Branching-Strategie zur Kombination mit Ihren DevOps Prozessen ist entscheidend für den Erfolg. Wenn Sie die richtige Git-Branching-Strategie für Ihr Unternehmen auswählen, können Sie DevOps Standards und Best Practices zwischen den Entwicklungsteams präzise kommunizieren. Git-Branching kann in einer einzigen Umgebung einfach sein, aber es kann verwirrend werden, wenn es auf mehrere Umgebungen angewendet wird, z. B. in Sandbox-, Entwicklungs-, Test-, Staging- und Produktionsumgebungen. Wenn mehrere Umgebungen vorhanden sind, erhöht sich die Komplexität der Implementierung. DevOps

Dieser Leitfaden enthält visuelle Diagramme von Git-Branching-Strategien, die zeigen, wie ein Unternehmen einen DevOps Multi-Account-Prozess implementieren kann. Visuelle Anleitungen helfen Teams zu verstehen, wie sie ihre Git-Branching-Strategien mit ihren DevOps Praktiken verbinden können. Die Verwendung eines Standard-Branching-Modells wie Gitflow, GitHub Flow oder Trunk für die Verwaltung des Quellcode-Repositorys hilft Entwicklungsteams, ihre Arbeit besser aufeinander abzustimmen. Diese Teams können auch Standard-Git-Schulungsressourcen im Internet nutzen, um diese Modelle und Strategien zu verstehen und umzusetzen.

DevOps Bewährte Verfahren dazu AWS finden Sie in den DevOpsLeitlinien in AWS Well-Architected. Gehen Sie bei der Lektüre dieses Leitfadens sorgfältig vor, um die richtige Filialstrategie für Ihr Unternehmen auszuwählen. Einige Strategien passen möglicherweise besser zu Ihrem Anwendungsfall als andere.

Ziele

Dieser Leitfaden ist Teil einer Dokumentationsreihe über die Auswahl und Implementierung von DevOps Branching-Strategien für Unternehmen mit mehreren AWS-Konten. Diese Reihe soll Ihnen helfen, von Anfang an die Strategie anzuwenden, die Ihren Anforderungen, Zielen und bewährten

Verfahren am besten entspricht, um Ihre Erfahrung in der AWS Cloud Dieses Handbuch enthält keine DevOps ausführbaren Skripts, da sie je nach der CI/CD-Engine (Continuous Integration and Continuous Delivery) und den Technologie-Frameworks, die Ihr Unternehmen verwendet, variieren.

In diesem Leitfaden werden die Unterschiede zwischen drei gängigen Git-Branching-Strategien erklärt: GitHub Flow, Gitflow und Trunk. Die Empfehlungen in diesem Leitfaden helfen Teams dabei, eine Branching-Strategie zu finden, die ihren Unternehmenszielen entspricht. Nachdem Sie diesen Leitfaden gelesen haben, sollten Sie in der Lage sein, eine Branching-Strategie für Ihr Unternehmen auszuwählen. Nachdem Sie sich für eine Strategie entschieden haben, können Sie eines der folgenden Muster verwenden, um diese Strategie gemeinsam mit Ihren Entwicklungsteams umzusetzen:

- Implementieren Sie eine Trunk-Branching-Strategie für Umgebungen mit mehreren Konten DevOps
- Implementieren Sie eine GitHub Flow-Branching-Strategie für Umgebungen mit mehreren Konten DevOps
- Implementieren Sie eine Gitflow-Branching-Strategie für Umgebungen mit mehreren Konten DevOps

Es ist wichtig zu beachten, dass das, was für eine Organisation, ein Team oder ein Projekt funktioniert, möglicherweise nicht für andere geeignet ist. Die Wahl zwischen Git-Branching-Strategien hängt von verschiedenen Faktoren ab, wie der Teamgröße, den Projektanforderungen und dem gewünschten Gleichgewicht zwischen Zusammenarbeit, Integrationshäufigkeit und Release-Management.

Verwenden von Praktiken CI/CD

AWS empfiehlt die Implementierung von Continuous Integration und Continuous Delivery (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/CDPipelines dienen auch als Steuerung und Leitplanken für Entwicklungsteams). Sie setzen Konsistenz, Standards, bewährte Verfahren und Mindestakzeptanzniveaus für die Akzeptanz und Bereitstellung von Funktionen durch. Weitere Informationen finden Sie unter Practicing Continuous Integration and Continuous Delivery auf AWS.

Alle in diesem Leitfaden erörterten Branching-Strategien eignen sich hervorragend dazu, 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 Leitplanken und Mechanismen für Entwicklungsteams festzulegen, die verhindern, dass Entwickler den definierten Prozess absichtlich oder unabsichtlich umgehen.

AWS bietet eine Reihe von Entwicklerservices, die Sie beim Aufbau von CI/CD-Pipelines unterstützen sollen. AWS CodePipelineIst beispielsweise ein vollständig verwalteter Continuous Delivery Service, der Sie bei der Automatisierung Ihrer Release-Pipelines für schnelle und zuverlässige Anwendungs- und Infrastrukturupdates unterstützt. AWS CodeBuildkompiliert Quellcode, führt Tests durch und produziert ready-to-deploy Softwarepakete. Weitere Informationen finden Sie unter Entwicklertools unter AWS.

Die DevOps Umgebungen verstehen

Um die Branching-Strategien zu verstehen, müssen Sie den Zweck und die Aktivitäten verstehen, die in jeder Umgebung stattfinden. Durch die Einrichtung mehrerer Umgebungen können Sie die Entwicklungsaktivitäten in Phasen unterteilen, diese Aktivitäten überwachen und die unbeabsichtigte Veröffentlichung nicht genehmigter Funktionen verhindern. In jeder Umgebung können Sie eine oder mehrere AWS-Konten haben.

Die meisten Organisationen verfügen über mehrere Umgebungen, die für den Einsatz vorgesehen sind. Die Anzahl der Umgebungen kann jedoch je nach Organisation und Softwareentwicklungsrichtlinien variieren. In dieser Dokumentationsserie wird davon ausgegangen, dass Sie über die folgenden fünf gängigen Umgebungen verfügen, die sich über Ihre Entwicklungspipeline erstrecken, obwohl sie möglicherweise unterschiedliche Namen haben:

- Sandbox — Eine Umgebung, in der Entwickler Code schreiben, Fehler machen und Machbarkeitsstudien durchführen.
- Entwicklung — Eine Umgebung, in der Entwickler ihren Code integrieren, um sicherzustellen, dass alles als eine einzige, zusammenhängende Anwendung funktioniert.
- Testen — Eine Umgebung, in der QA-Teams oder Abnahmetests stattfinden. Teams führen in dieser Umgebung häufig Leistungs- oder Integrationstests durch.
- Staging — Eine Vorproduktionsumgebung, in der Sie überprüfen, ob der Code und die Infrastruktur unter produktionsäquivalenten Umständen erwartungsgemäß funktionieren. Diese Umgebung ist so konfiguriert, dass sie der Produktionsumgebung so ähnlich wie möglich ist.
- Produktion — Eine Umgebung, die den Datenverkehr Ihrer Endbenutzer und Kunden verarbeitet.

In diesem Abschnitt werden die einzelnen Umgebungen detailliert beschrieben. Außerdem werden die Build-Schritte, Bereitstellungsschritte und Exit-Kriterien für jede Umgebung beschrieben, sodass Sie mit der nächsten fortfahren können. Die folgende Abbildung zeigt diese Umgebungen nacheinander.

Themen in diesem Abschnitt:

Sandbox-Umgebung

In der Sandbox-Umgebung schreiben Entwickler Code, machen Fehler und führen Machbarkeitsstudien durch. Sie können die Bereitstellung in einer Sandbox-Umgebung von einer lokalen Arbeitsstation aus oder über ein Skript auf einer lokalen Arbeitsstation durchführen.

Zugriff

Entwickler sollten vollen Zugriff auf die Sandbox-Umgebung haben.

Schritte erstellen

Entwickler führen den Build manuell auf ihren lokalen Workstations aus, wenn sie bereit sind, Änderungen an der Sandbox-Umgebung vorzunehmen.

1. Verwenden Sie git-secrets (GitHub), um nach vertraulichen Informationen zu suchen
2. Lint den Quellcode
3. Erstellen und kompilieren Sie den Quellcode, falls zutreffend
4. Führen Sie Komponententests durch
5. Führen Sie eine Analyse der Codeabdeckung durch
6. Eine statische Codeanalyse durchführen
7. Erstellen Sie Infrastruktur als Code (IaC)
8. Führen Sie eine IaC-Sicherheitsanalyse durch
9. Extrahieren Sie Open-Source-Lizenzen
10. Veröffentlichen Sie Build-Artefakte

Schritte zur Bereitstellung

Wenn Sie die Modelle Gitflow oder Trunk verwenden, werden die Bereitstellungsschritte automatisch eingeleitet, wenn ein `feature` Branch erfolgreich in der Sandbox-Umgebung erstellt wurde. Wenn Sie das GitHub Flow-Modell verwenden, führen Sie die folgenden Bereitstellungsschritte manuell aus. Im Folgenden sind die Bereitstellungsschritte in der Sandbox-Umgebung aufgeführt:

1. Laden Sie veröffentlichte Artefakte herunter
2. Führen Sie die Datenbank-Versionierung durch
3. Führen Sie die IaC-Bereitstellung durch
4. Führen Sie Integrationstests durch

Erwartungen vor dem Übergang zur Entwicklungsumgebung

- Erfolgreicher Aufbau der `feature` Filiale in der Sandbox-Umgebung
- Ein Entwickler hat die Funktion manuell in der Sandbox-Umgebung bereitgestellt und getestet

Entwicklungsumgebung

In der Entwicklungsumgebung integrieren Entwickler ihren Code zusammen, um sicherzustellen, dass alles als eine zusammenhängende Anwendung funktioniert. In Gitflow enthält die Entwicklungsumgebung die neuesten Funktionen, die in der Merge-Anfrage enthalten sind und zur Veröffentlichung bereit sind. In GitHub Flow- und Trunk-Strategien wird die Entwicklungsumgebung als Testumgebung betrachtet, und die Codebasis ist möglicherweise instabil und für die Bereitstellung in der Produktion ungeeignet.

Zugriff

Weisen Sie Berechtigungen nach dem Prinzip der geringsten Rechte zu. Geringste Berechtigung ist die bewährte Methode, die für die Ausführung einer Aufgabe erforderlichen Mindestberechtigungen zu gewähren. Entwickler sollten weniger Zugriff auf die Entwicklungsumgebung als auf die Sandbox-Umgebung haben.

Schritte erstellen

Wenn Sie eine Merge-Anfrage für den `develop` Branch (Gitflow) oder den `main` Branch (Trunk oder GitHub Flow) erstellen, wird der Build automatisch gestartet.

1. Verwenden Sie git-secrets (GitHub), um nach vertraulichen Informationen zu suchen
2. Lint den Quellcode
3. Erstellen und kompilieren Sie den Quellcode, falls zutreffend
4. Führen Sie Komponententests durch
5. Führen Sie eine Analyse der Codeabdeckung durch
6. Eine statische Codeanalyse durchführen
7. IaC erstellen
8. Führen Sie eine IaC-Sicherheitsanalyse durch
9. Extrahieren Sie Open-Source-Lizenzen

Schritte zur Bereitstellung

Wenn Sie das Gitflow-Modell verwenden, werden die Bereitstellungsschritte automatisch eingeleitet, wenn ein `develop` Branch erfolgreich in der Entwicklungsumgebung erstellt wurde. Wenn du das GitHub Flow-Modell oder das Trunk-Modell verwendest, werden die Bereitstellungsschritte automatisch eingeleitet, wenn eine Merge-Anfrage für den `main` Branch erstellt wird. Im Folgenden sind die Bereitstellungsschritte in der Entwicklungsumgebung aufgeführt:

1. Laden Sie die veröffentlichten Artefakte aus den Build-Schritten herunter
2. Führen Sie die Datenbank-Versionierung durch
3. Führen Sie die IaC-Bereitstellung durch
4. Führen Sie Integrationstests durch

Erwartungen vor dem Übergang zur Testumgebung

- Erfolgreicher Aufbau und Bereitstellung des `develop` Branches (Gitflow) oder des `main` Branches (Trunk oder GitHub Flow) in der Entwicklungsumgebung
- Der Komponententest besteht zu 100%

- Erfolgreicher IaC-Build
- Bereitstellungsartefakte wurden erfolgreich erstellt
- Ein Entwickler hat eine manuelle Überprüfung durchgeführt, um sicherzustellen, dass die Funktion wie erwartet funktioniert

Testumgebung

Mitarbeiter der Qualitätssicherung (QA) verwenden die Testumgebung, um Funktionen zu validieren. Sie genehmigen die Änderungen, nachdem sie die Tests abgeschlossen haben. Nach der Genehmigung geht die Filiale zur nächsten Umgebung über, dem Staging. In Gitflow sind diese und andere Umgebungen darüber nur für die Bereitstellung in Branches verfügbar. `release` Ein `release` Branch basiert auf einem `develop` Branch, der die geplanten Funktionen enthält.

Zugriff

Weisen Sie Berechtigungen nach dem Prinzip der geringsten Rechte zu. Entwickler sollten weniger Zugriff auf die Testumgebung als auf die Entwicklungsumgebung haben. Das QA-Personal benötigt ausreichende Berechtigungen, um die Funktion zu testen.

Schritte erstellen

Der Build-Prozess in dieser Umgebung gilt nur für Bugfixes, wenn die Gitflow-Strategie verwendet wird. Wenn Sie eine Merge-Anfrage für den `bugfix` Branch erstellen, wird der Build automatisch gestartet.

1. Verwenden Sie git-secrets (GitHub), um nach vertraulichen Informationen zu suchen
2. Lint den Quellcode
3. Erstellen und kompilieren Sie den Quellcode, falls zutreffend
4. Führen Sie Komponententests durch
5. Führen Sie eine Analyse der Codeabdeckung durch
6. Eine statische Codeanalyse durchführen
7. IaC erstellen
8. Führen Sie eine IaC-Sicherheitsanalyse durch
9. Extrahieren Sie Open-Source-Lizenzen

Schritte zur Bereitstellung

Initiieren Sie nach der `release` Bereitstellung in der Entwicklungsumgebung automatisch die Bereitstellung des `main` Branches (GitHub Gitflow) oder des Branches (Trunk oder Flow) in der Testumgebung. Im Folgenden sind die Bereitstellungsschritte in der Testumgebung aufgeführt:

1. Stellen Sie den `release` Branch (Gitflow) oder `main` Branch (Trunk oder GitHub Flow) in der Testumgebung bereit
2. Machen Sie eine Pause für die manuelle Genehmigung durch das dafür vorgesehene Personal
3. Laden Sie veröffentlichte Artefakte herunter
4. Führen Sie die Datenbank-Versionierung durch
5. Führen Sie die IaC-Bereitstellung durch
6. Führen Sie Integrationstests durch
7. Führen Sie Leistungstests durch
8. Zulassung zur Qualitätssicherung

Erwartungen vor dem Übergang zur Staging-Umgebung

- Die Entwicklungs- und QA-Teams haben ausreichend Tests durchgeführt, um die Anforderungen Ihres Unternehmens zu erfüllen.
- Das Entwicklungsteam hat alle entdeckten Fehler über eine `bugfix` Filiale behoben.

Staging-Umgebung

Die Staging-Umgebung ist so konfiguriert, dass sie der Produktionsumgebung entspricht. Beispielsweise sollte das Daten-Setup in Umfang und Größe den Produktionsworkloads ähneln. Verwenden Sie die Staging-Umgebung, um zu überprüfen, ob Code und Infrastruktur erwartungsgemäß funktionieren. Diese Umgebung ist auch die bevorzugte Wahl für geschäftliche Anwendungsfälle wie Vorschauen oder Kundenvorführungen.

Zugriff

Weisen Sie Berechtigungen nach dem Prinzip der geringsten Rechte zu. Entwickler sollten denselben Zugriff auf die Staging-Umgebung haben wie auf die Produktionsumgebung.

Schritte erstellen

Keine. Dieselben Artefakte, die in der Testumgebung verwendet wurden, werden in der Staging-Umgebung wiederverwendet.

Schritte zur Bereitstellung

Initiieren Sie nach der `release` Genehmigung und Bereitstellung in der Testumgebung automatisch die Bereitstellung des `main` Branches (GitHub Gitflow) oder des Branches (Trunk oder Flow) in der Staging-Umgebung. Im Folgenden sind die Bereitstellungsschritte in der Staging-Umgebung aufgeführt:

1. Stellen Sie den `release` Branch (Gitflow) oder `main` Branch (Trunk oder GitHub Flow) in der Staging-Umgebung bereit
2. Machen Sie eine Pause für die manuelle Genehmigung durch das dafür vorgesehene Personal
3. Laden Sie veröffentlichte Artefakte herunter
4. Führen Sie die Datenbank-Versionierung durch
5. Führen Sie die IaC-Bereitstellung durch
6. (Optional) Führen Sie Integrationstests durch
7. (Optional) Führen Sie Belastungstests durch
8. Holen Sie sich die Genehmigung der für die Entwicklung, Qualitätssicherung, Produkte oder Geschäftsbereiche zuständigen Stelle ein

Erwartungen vor dem Übergang zur Produktionsumgebung

- Eine produktionsäquivalente Version wurde erfolgreich in der Staging-Umgebung bereitgestellt
- (Optional) Integration und Belastungstests waren erfolgreich

Produktionsumgebung

Die Produktionsumgebung unterstützt das veröffentlichte Produkt und verarbeitet echte Daten von echten Kunden. Dabei handelt es sich um eine geschützte Umgebung, der der Zugriff nach den geringsten Rechten zugewiesen wird. Der Zugriff mit erhöhten Rechten sollte nur im Rahmen eines geprüften Ausnahmeprozesses für einen begrenzten Zeitraum gewährt werden.

Zugriff

In der Produktionsumgebung sollten Entwickler eingeschränkten Lesezugriff auf die AWS-Managementkonsole haben. Entwickler sollten beispielsweise für den täglichen Betrieb auf Protokolldaten zugreifen können. Alle Produktionsversionen sollten vor der Bereitstellung einem Genehmigungsschritt unterzogen werden.

Schritte erstellen

Keine. Dieselben Artefakte, die in den Test- und Staging-Umgebungen verwendet wurden, werden in der Produktionsumgebung wiederverwendet.

Schritte zur Bereitstellung

Initiieren Sie nach der Genehmigung und Bereitstellung in der Staging-Umgebung automatisch die Bereitstellung der `release` `main` Filiale (GitHub Gitflow) oder der Filiale (Trunk oder Flow) in der Produktionsumgebung. Im Folgenden sind die Bereitstellungsschritte in der Produktionsumgebung aufgeführt:

1. Stellen Sie den `release` Branch (Gitflow) oder `main` Branch (Trunk oder GitHub Flow) in der Produktionsumgebung bereit
2. Machen Sie eine Pause für die manuelle Genehmigung durch das dafür vorgesehene Personal
3. Laden Sie veröffentlichte Artefakte herunter
4. Führen Sie die Datenbank-Versionierung durch
5. Führen Sie die IaC-Bereitstellung durch

Bewährte Methoden für die Git-basierte Entwicklung

Um die Git-basierte Entwicklung erfolgreich einzuführen, ist es wichtig, eine Reihe von Best Practices zu befolgen, die die Zusammenarbeit fördern, die Codequalität aufrechterhalten und Continuous Integration und Continuous Delivery (CI/CD) unterstützen. Lesen Sie zusätzlich zu den bewährten Methoden in diesem Leitfaden auch die AWS Well-Architected DevOps Guidance. Im Folgenden finden Sie einige der wichtigsten Best Practices für die Git-basierte Entwicklung auf AWS:

- Halten Sie die Änderungen klein und häufig — Ermutigen Sie Entwickler, kleine, schrittweise Änderungen oder Funktionen vorzunehmen. Dies reduziert das Risiko von Zusammenführungskonflikten und erleichtert die schnelle Identifizierung und Behebung von Problemen.
- Funktionsumschalter verwenden — Verwenden Sie Funktionsumschalter oder Feature-Flags, um die Veröffentlichung unvollständiger oder experimenteller Funktionen zu verwalten. Auf diese Weise können Sie bestimmte Funktionen in der Produktion ausblenden, aktivieren oder deaktivieren, ohne die Stabilität des Hauptzweigs zu beeinträchtigen.
- Sorgen Sie für eine robuste Testsuite — Eine umfassende, gut gepflegte Testsuite ist entscheidend, um Probleme frühzeitig zu erkennen und zu überprüfen, ob die Codebasis stabil bleibt. Investieren Sie in die Testautomatisierung und priorisieren Sie die Behebung fehlgeschlagener Tests.
- Setzen Sie auf kontinuierliche Integration — Verwenden Sie Tools und Methoden für die kontinuierliche Integration, um Codeänderungen automatisch zu erstellen, zu testen und in den `develop` Branch (Gitflow) oder `main` Branch (Trunk oder GitHub Flow) zu integrieren. Dies hilft Ihnen, Probleme frühzeitig zu erkennen und den Entwicklungsprozess zu optimieren.
- Führen Sie Code-Reviews durch — Fördern Sie Peer-Reviews von Code, um die Qualität aufrechtzuerhalten, Wissen catch und potenzielle Probleme zu erkennen, bevor sie in die `main` Branche integriert werden. Verwenden Sie Pull-Requests oder andere Code-Review-Tools, um diesen Prozess zu vereinfachen.
- Überwachen und reparieren Sie fehlerhafte Builds — Wenn ein Build kaputt geht oder Tests fehlschlagen, sollten Sie der schnellstmöglichen Behebung des Problems Priorität einräumen. Dadurch bleibt der `develop` Branch (Gitflow) oder `main` Branch (Trunk oder GitHub Flow) in einem Zustand, der veröffentlicht werden kann, und die Auswirkungen auf andere Entwickler werden minimiert.

- Kommunizieren und zusammenarbeiten — Fördern Sie die offene Kommunikation und Zusammenarbeit zwischen den Teammitgliedern. Stellen Sie sicher, dass Entwickler über die laufenden Arbeiten und Änderungen an der Codebasis informiert sind.
- Kontinuierliches Refaktorieren — Refaktorieren Sie die Codebasis regelmäßig, um ihre Wartbarkeit zu verbessern und technische Schulden zu reduzieren. Ermutigen Sie Entwickler, den Code in einem besseren Zustand zu belassen, als sie ihn vorgefunden haben.
- Verwenden Sie kurzlebige Branches für komplexe Aufgaben — Verwenden Sie für größere oder komplexere Aufgaben kurzlebige Branches (auch als Task-Branches bezeichnet), um an den Änderungen zu arbeiten. Achten Sie jedoch darauf, die Lebensdauer der Filialen kurz zu halten, in der Regel weniger als einen Tag. Führen Sie die Änderungen so schnell wie möglich wieder in den `develop` Branch (Gitflow) oder `main` Branch (Trunk oder GitHub Flow) ein. Kleinere und häufigere Zusammenführungen und Überprüfungen sind für ein Team einfacher zu verarbeiten und zu verarbeiten als eine große Zusammenführungsanfrage.
- Schulung und Unterstützung des Teams — Bieten Sie Schulungen und Support für Entwickler an, die noch keine Erfahrung mit der Git-basierten Entwicklung haben oder Unterstützung bei der Übernahme ihrer Best Practices benötigen.

Strategien zur Git-Verzweigung

In dieser Anleitung werden die folgenden Git-based Branching-Strategien, geordnet nach der geringsten bis zur komplexesten, detailliert beschrieben:

- Trunk-based Trunk-Entwicklung ist eine Softwareentwicklungspraxis, bei der alle Entwickler an einem einzigen Zweig arbeiten, der `trunk` üblicherweise als `main` Oder-Zweig bezeichnet wird. Die Idee hinter diesem Ansatz besteht darin, die Codebasis in einem kontinuierlich veröffentlichbaren Zustand zu halten, indem häufig Codeänderungen integriert und auf automatisierte Tests und kontinuierliche Integration zurückgegriffen wird.
- GitHub Flow — GitHub Flow ist ein leichter, branchenbasierter Workflow, der von entwickelt wurde. GitHub Er basiert auf der Idee kurzlebiger `feature` Filialen. Wenn ein Feature fertig und bereit für die Bereitstellung ist, wird es mit dem `main` Branch zusammengeführt.
- Gitflow — Bei einem Gitflow-Ansatz wird die Entwicklung in einzelnen Feature-Branches abgeschlossen. Nach der Genehmigung fügst du `feature` Branches zu einem Integrationszweig zusammen, der normalerweise einen Namen trägt. `develop` Wenn sich in dem `develop` Zweig genügend Funktionen angesammelt haben, wird ein `release` Zweig erstellt, um die Funktionen in höheren Umgebungen bereitzustellen.

Jede Branching-Strategie hat Vor- und Nachteile. Sie verwenden zwar alle dieselben Umgebungen, aber nicht alle verwenden dieselben Branches oder manuellen Genehmigungsschritte. Sehen Sie sich in diesem Abschnitt des Leitfadens jede Branching-Strategie im Detail an, sodass Sie mit ihren Nuancen vertraut sind und beurteilen können, ob sie für den Anwendungsfall Ihres Unternehmens geeignet ist.

Themen in diesem Abschnitt:

Strategie zur Stammverzweigung

Trunk-based Entwicklung ist eine Softwareentwicklungspraxis, bei der alle Entwickler an einem einzigen Zweig arbeiten, der üblicherweise als `trunk` `main` Oder-Zweig bezeichnet wird. Die Idee

hinter diesem Ansatz besteht darin, die Codebasis in einem kontinuierlich veröffentlichbaren Zustand zu halten, indem häufig Codeänderungen integriert und auf automatisierte Tests und kontinuierliche Integration zurückgegriffen wird.

Bei der Trunk-basierten Entwicklung übertragen Entwickler ihre Änderungen mehrmals täglich in den `main` Branch, wobei kleine, inkrementelle Updates angestrebt werden. Dies ermöglicht schnelle Feedback-Schleifen, reduziert das Risiko von Zusammenführungskonflikten und fördert die Zusammenarbeit zwischen den Teammitgliedern. In dieser Praxis wird die Bedeutung einer gut gepflegten Testsuite betont, da sie auf automatisierten Tests basiert, um potenzielle Probleme frühzeitig zu catch und sicherzustellen, dass die Codebasis stabil und veröffentlichbar bleibt.

Trunk-based Entwicklung wird häufig mit funktionsbasierter Entwicklung (auch bekannt als Feature Branching oder Feature-Driven Development) verglichen, bei der jedes neue Feature oder jeder Bugfix in einem eigenen, vom Hauptzweig getrennten Zweig entwickelt wird. Die Wahl zwischen Trunk-basierter Entwicklung und Feature-basierter Entwicklung hängt von Faktoren wie Teamgröße, Projektanforderungen und dem gewünschten Gleichgewicht zwischen Zusammenarbeit, Integrationshäufigkeit und Release-Management ab.

Weitere Informationen zur Trunk-Branching-Strategie finden Sie in den folgenden Ressourcen:

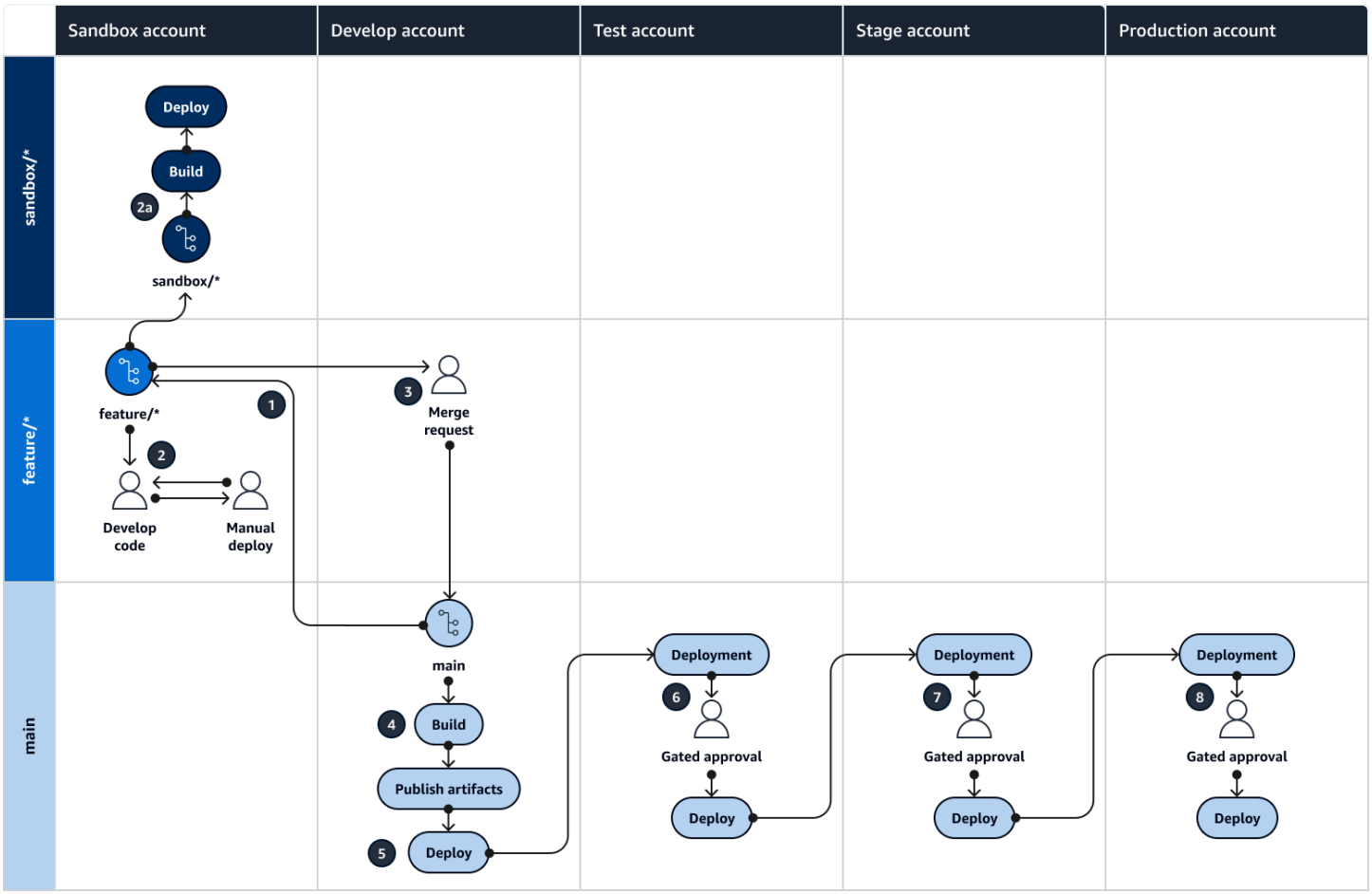
- Implementieren Sie eine Trunk-Branching-Strategie für DevOps Umgebungen mit mehreren Konten (AWS Prescriptive Guidance)
- Einführung in die Trunk-Based Entwicklung (Website zur trunkbasierten Entwicklung)

Themen in diesem Abschnitt:

Visueller Überblick über die Trunk-Strategie

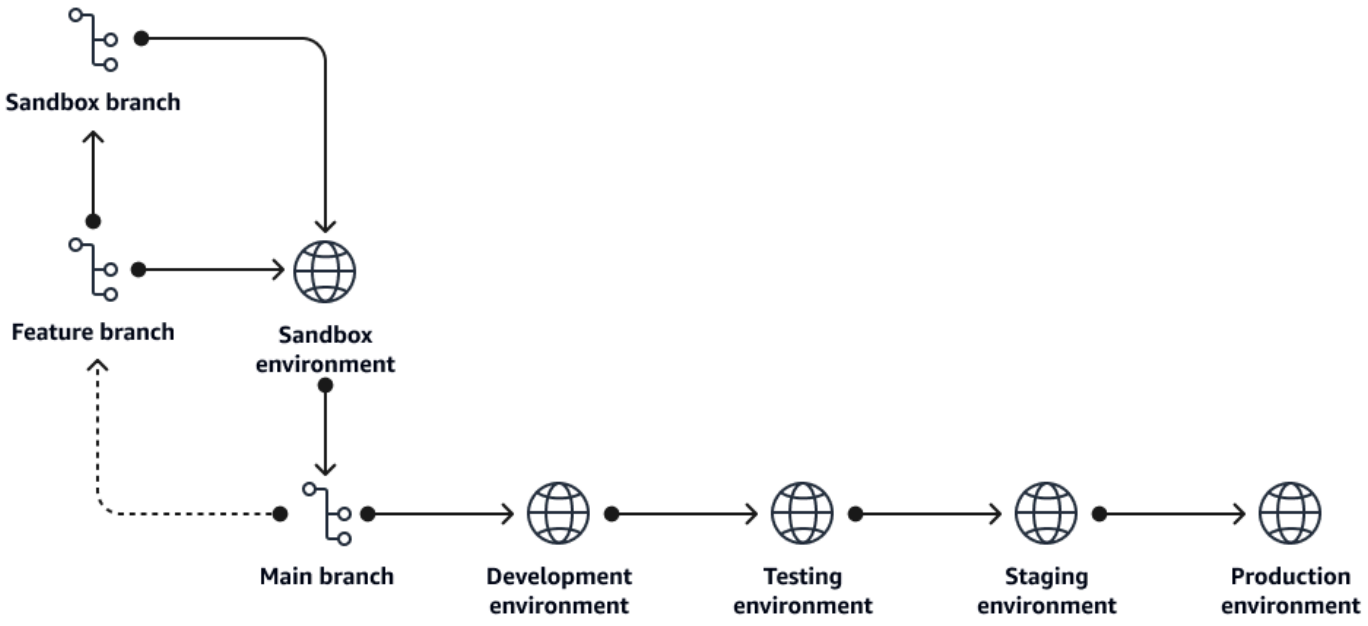
Das folgende Diagramm kann wie ein Punnett-Quadrat (Wikipedia) verwendet werden, um die Trunk-Branching-Strategie zu verstehen. Richten Sie die Zweige auf der vertikalen Achse mit den AWS Umgebungen auf der horizontalen Achse aus, um zu bestimmen, welche Aktionen in den einzelnen Szenarien ausgeführt werden sollen. Die eingekreisten Zahlen führen Sie durch die Reihenfolge der Aktionen, die im Diagramm dargestellt sind. Dieses Diagramm zeigt den Entwicklungsablauf

einer Trunk-Branching-Strategie von einem `feature` Branch in der Sandbox-Umgebung bis zur Produktionsversion des Branches. `main` Weitere Informationen zu den Aktivitäten, die in den einzelnen Umgebungen stattfinden, finden Sie in diesem DevOps Handbuch unter Umgebungen.



Filialen in einer Trunk-Strategie

Eine Trunk-Branching-Strategie umfasst üblicherweise die folgenden Verzweigungen.



Feature-Zweig

Sie entwickeln Funktionen oder erstellen einen Hotfix in einer `feature` Filiale. Um einen `feature` Zweig zu erstellen, zweigen Sie von diesem `main` Zweig ab. Entwickler iterieren, übertragen und testen Code in einem `feature` Branch. Wenn ein Feature fertig ist, bewirbt der Entwickler das Feature. Von einem `feature` Zweig aus gibt es nur zwei Pfade vorwärts:

- Mit dem `sandbox` Zweig verschmelzen
- Erstellen Sie eine Anfrage zur Zusammenführung mit der `main` Filiale

Benennungskonvention:	`feature/<story number>_<developer initials>_<descriptor>`
Beispiel für eine Namenskonvention:	`feature/123456_MS_Implement_Feature_A`

Sandbox-Zweig

Bei diesem Zweig handelt es sich um einen nicht standardmäßigen Trunk-Zweig, der jedoch für die CI/CD Pipeline-Entwicklung nützlich ist. Der `sandbox` Zweig wird hauptsächlich für die folgenden Zwecke verwendet:

- Führen Sie mithilfe der Pipelines eine vollständige Bereitstellung in der Sandbox-Umgebung durch CI/CD
- Entwickeln und testen Sie eine Pipeline, bevor Sie Mergeanfragen für vollständige Tests in einer niedrigeren Umgebung, z. B. Entwicklung oder Testen, einreichen.

`Sandbox`Verzweigungen sind temporärer Natur und sollen kurzlebig sein. Sie sollten nach Abschluss der spezifischen Tests gelöscht werden.

Benennungskonvention:	`sandbox/<story number>_<developer initials>_<descriptor>`
Beispiel für eine Namenskonvention:	`sandbox/123456_MS_Test_Pipeline_Deploy`

Hauptzweig

Der `main` Zweig steht immer für den Code, der in der Produktion ausgeführt wird. Code wird abgezweigt`main`, entwickelt und anschließend wieder zusammengeführt. `main` Bereitstellungen von `main` können auf jede Umgebung abzielen. Um vor dem Löschen zu schützen, aktivieren Sie den Branch-Schutz für den `main` Branch.

Benennungskonvention:	`main`

Hotfix-Zweig

In einem Stamm-basierten Workflow gibt es keinen speziellen `hotfix` Branch. Hotfixes verwenden Branches. `feature`

Vor- und Nachteile der Trunk-Strategie

Die Trunk-Branching-Strategie eignet sich gut für kleinere, ausgereifte Entwicklungsteams mit ausgeprägten Kommunikationsfähigkeiten. Sie funktioniert auch gut, wenn Sie fortlaufende, fortlaufende Feature-Releases für die Anwendung haben. Es ist nicht gut geeignet, wenn Sie große oder fragmentierte Entwicklungsteams haben oder wenn Sie umfangreiche, geplante Feature-

Releases haben. Bei diesem Modell treten Zusammenführungskonflikte auf. Seien Sie sich also bewusst, dass die Lösung von Zusammenführungskonflikten eine Schlüsselkompetenz ist. Alle Teammitglieder müssen entsprechend geschult werden.

Vorteile

Trunk-based Die Entwicklung bietet mehrere Vorteile, die den Entwicklungsprozess verbessern, die Zusammenarbeit rationalisieren und die Gesamtqualität der Software verbessern können. Im Folgenden sind einige der wichtigsten Vorteile aufgeführt:

- Schnellere Feedback-Schleifen — Bei der Trunk-basierten Entwicklung integrieren Entwickler ihre Codeänderungen häufig, oft mehrmals täglich. Dies ermöglicht schnelleres Feedback zu potenziellen Problemen und hilft Entwicklern, Probleme schneller zu identifizieren und zu beheben, als dies bei einem funktionsbasierten Entwicklungsmodell der Fall wäre.
- Weniger Zusammenführungskonflikte — Bei der trunkbasierten Entwicklung wird das Risiko großer, komplizierter Zusammenführungskonflikte minimiert, da Änderungen kontinuierlich integriert werden. Dies trägt zur Aufrechterhaltung einer übersichtlicheren Codebasis bei und reduziert den Zeitaufwand für die Lösung von Konflikten. Das Lösen von Konflikten kann bei der funktionsbasierten Entwicklung sowohl zeitaufwändig als auch fehleranfällig sein.
- Verbesserte Zusammenarbeit — Die Trunk-based Entwicklung ermutigt Entwickler, in derselben Branche zusammenzuarbeiten, was zu einer besseren Kommunikation und Zusammenarbeit innerhalb des Teams führt. Dies kann zu einer schnelleren Problemlösung und einer kohärenteren Teamdynamik führen.
- Einfachere Code-Reviews — Da Codeänderungen bei der Trunk-basierten Entwicklung kleiner und häufiger sind, kann es einfacher sein, gründliche Code-Reviews durchzuführen. Kleinere Änderungen sind im Allgemeinen leichter zu verstehen und zu überprüfen, was zu einer effektiveren Identifizierung potenzieller Probleme und Verbesserungen führt.
- Kontinuierliche Integration und Bereitstellung — Die Trunk-based Entwicklung unterstützt die Prinzipien der kontinuierlichen Integration und kontinuierlichen Bereitstellung (CI/CD). Indem die Codebasis in einem veröffentlichbaren Zustand gehalten und Änderungen häufig integriert werden, können Teams leichter CI/CD Praktiken anwenden, was zu schnelleren Bereitstellungszyklen und verbesserter Softwarequalität führt.
- Verbesserte Codequalität — Durch häufige Integrationen, Tests und Codeüberprüfungen kann die Trunk-basierte Entwicklung zu einer insgesamt besseren Codequalität beitragen. Entwickler können Probleme schneller catch und beheben und so die Wahrscheinlichkeit verringern, dass sich im Laufe der Zeit technische Schulden ansammeln.

- Vereinfachte Branching-Strategie — Die Trunk-based Entwicklung vereinfacht die Branching-Strategie, indem die Anzahl langlebiger Filialen reduziert wird. Dies kann die Verwaltung und Wartung der Codebasis erleichtern, insbesondere bei großen Projekten oder Teams.

Nachteile

Trunk-based Die Entwicklung hat einige Nachteile, die sich auf den Entwicklungsprozess und die Teamdynamik auswirken können. Im Folgenden sind einige bemerkenswerte Nachteile aufgeführt:

- Eingeschränkte Isolierung — Da alle Entwickler in derselben Branche arbeiten, sind ihre Änderungen sofort für alle Teammitglieder sichtbar. Dies kann zu Störungen oder Konflikten führen, unbeabsichtigte Nebenwirkungen haben oder den Build beschädigen. Im Gegensatz dazu isoliert die funktionsbasierte Entwicklung Änderungen besser, sodass Entwickler unabhängiger arbeiten können.
- Erhöhter Testdruck — Die Trunk-based Entwicklung setzt auf kontinuierliche Integration und automatisierte Tests, um catch schnell zu erkennen. Dieser Ansatz kann jedoch großen Druck auf die Testinfrastruktur ausüben und erfordert eine gut gepflegte Testsuite. Wenn die Tests nicht umfassend oder zuverlässig sind, kann dies zu unentdeckten Problemen in der Hauptzweige führen.
- Weniger Kontrolle über Releases — die Trunk-based Entwicklung zielt darauf ab, die Codebasis in einem kontinuierlich veröffentlichbaren Zustand zu halten. Dies kann zwar von Vorteil sein, eignet sich aber möglicherweise nicht immer für Projekte mit strengen Veröffentlichungszeitplänen oder für Projekte, bei denen bestimmte Funktionen gemeinsam veröffentlicht werden müssen. Feature-basedDie Entwicklung bietet mehr Kontrolle darüber, wann und wie Funktionen veröffentlicht werden.
- Codefluktuation — Da Entwickler ständig Änderungen in den Hauptzweig integrieren, kann die Trunk-basierte Entwicklung zu einer erhöhten Codeabwanderung führen. Dies kann es Entwicklern erschweren, den Überblick über den aktuellen Stand der Codebasis zu behalten, und es kann zu Verwirrung führen, wenn versucht wird, die Auswirkungen der letzten Änderungen zu verstehen.
- Erfordert eine starke Teamkultur — Trunk-based Entwicklung erfordert ein hohes Maß an Disziplin, Kommunikation und Zusammenarbeit zwischen den Teammitgliedern. Es kann schwierig sein, dies aufrechtzuerhalten, insbesondere in größeren Teams oder bei der Zusammenarbeit mit Entwicklern, die mit diesem Ansatz weniger Erfahrung haben.
- Herausforderungen bei der Skalierbarkeit — Mit zunehmender Größe des Entwicklungsteams kann die Anzahl der Codeänderungen, die in den Hauptzweig integriert werden, schnell zunehmen. Dies

kann zu häufigeren Build-Unterbrechungen und Testfehlern führen, was es schwierig macht, die Codebasis in einem veröffentlichbaren Zustand zu halten.

GitHub Strategie zur Flow-Branchierung

GitHub Flow ist ein leichter, branchenbasierter Workflow, der von entwickelt wurde. GitHub GitHub Flow basiert auf der Idee kurzlebiger Feature-Branches, die mit dem Hauptzweig zusammengeführt werden, wenn das Feature fertig und einsatzbereit ist. Die wichtigsten Prinzipien von GitHub Flow sind:

- Branching ist leichtgewichtig — Entwickler können mit nur wenigen Klicks Feature-Branches für ihre Arbeit erstellen und so die Fähigkeit zur Zusammenarbeit und zum Experimentieren verbessern, ohne dass der Hauptzweig beeinträchtigt wird.
- Kontinuierliche Bereitstellung — Änderungen werden implementiert, sobald sie in den Hauptzweig integriert sind, was schnelles Feedback und schnelle Iterationen ermöglicht.
- Anfragen zusammenführen — Entwickler verwenden Merge-Anfragen, um einen Diskussions- und Überprüfungsprozess einzuleiten, bevor sie ihre Änderungen im Hauptzweig zusammenführen.

Weitere Informationen zu GitHub Flow finden Sie in den folgenden Ressourcen:

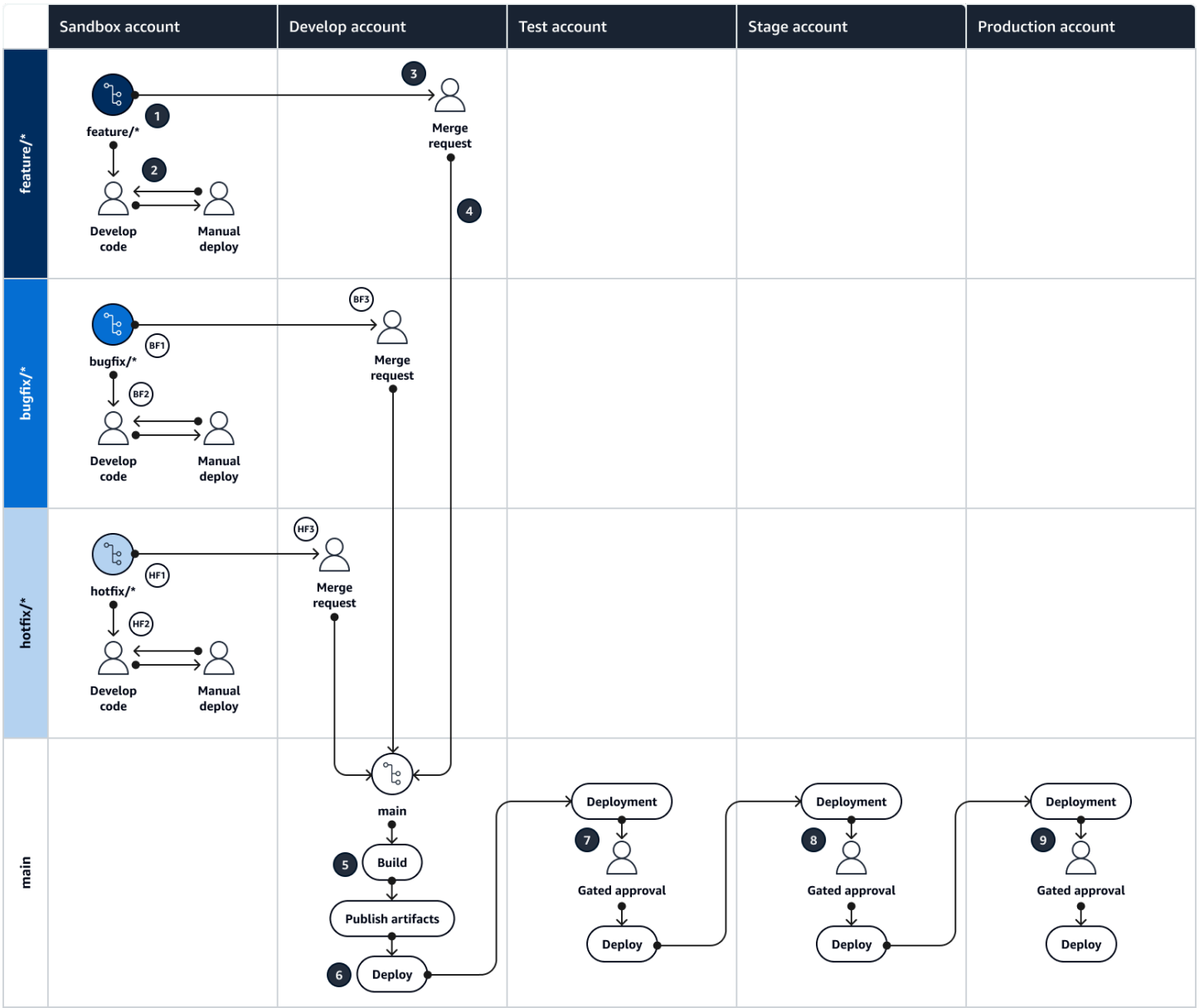
- Implementieren Sie eine GitHub Flow-Branching-Strategie für DevOps Umgebungen mit mehreren Konten (AWS Prescriptive Guidance)
- GitHub Flow Quickstart (Dokumentation) GitHub
- Warum GitHub Flow? (GitHubFlow-Webseite)

Themen in diesem Abschnitt:

Visueller Überblick über die GitHub Flow-Strategie

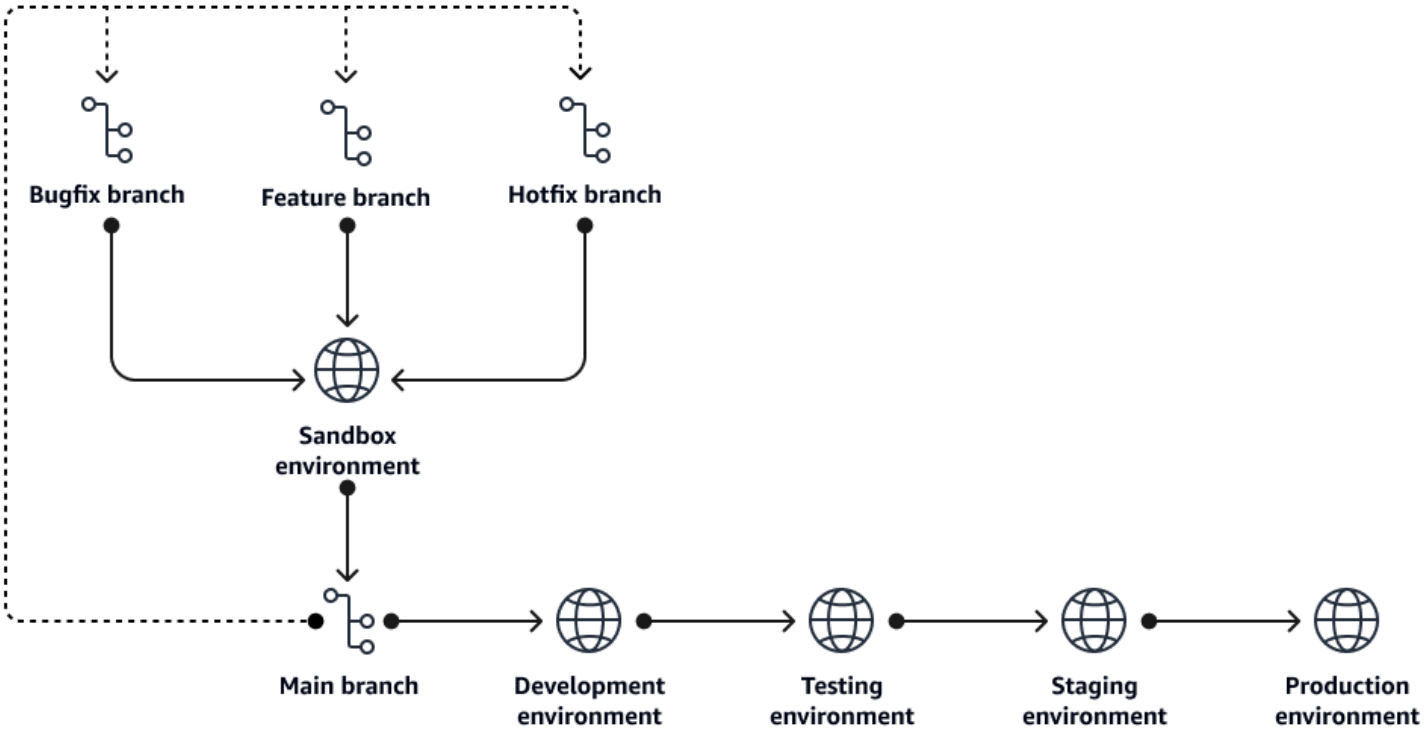
Das folgende Diagramm kann wie ein Punnett-Quadrat verwendet werden, um die GitHub Flow-Branching-Strategie zu verstehen. Richten Sie die Zweige auf der vertikalen Achse mit den AWS

Umgebungen auf der horizontalen Achse aus, um zu bestimmen, welche Aktionen in den einzelnen Szenarien ausgeführt werden sollen. Die eingekreisten Zahlen führen Sie durch die Reihenfolge der Aktionen, die im Diagramm dargestellt sind. Dieses Diagramm zeigt den GitHub Entwicklungsablauf einer Flow-Branching-Strategie von einem Feature-Branch in der Sandbox-Umgebung bis zur Produktionsversion des Hauptzweigs. Weitere Informationen zu den Aktivitäten, die in den einzelnen Umgebungen stattfinden, finden Sie in diesem DevOps Handbuch unter Umgebungen.



Filialen in einer GitHub Flow-Strategie

Eine GitHub Flow-Branching-Strategie besteht üblicherweise aus den folgenden Verzweigungen.



Feature-Zweig

Sie entwickeln Funktionen in `feature` Branchen. Um einen `feature` Zweig zu erstellen, zweigen Sie vom `main` Zweig ab. Entwickler iterieren, übernehmen und testen den Code im `feature` Branch. Wenn ein Feature fertiggestellt ist, bewirbt der Entwickler das Feature, indem er eine Merge-Anfrage an `main` erstellt.

Benennungskonvention:	`feature/<story number>_<developer initials>_<descriptor>`
Beispiel für eine Namenskonvention:	`feature/123456_MS_Implement_Feature_A`

Bugfix-Zweig

Der `bugfix` Branch wird verwendet, um Probleme zu beheben. Diese Zweige sind von der `main` Filiale abgezweigt. Nachdem der Bugfix in der Sandbox oder einer der niedrigeren Umgebungen getestet wurde, kann er auf höhere Umgebungen hochgestuft werden, indem er mit einer Merge-Anfrage zusammengeführt wird. `main` Dies ist eine vorgeschlagene Benennungskonvention für

Organisation und Nachverfolgung. Dieser Prozess könnte auch mithilfe eines Feature-Branches verwaltet werden.

Benennungskonvention:	`bugfix/<ticket number>_<developer initials>_<descriptor>`
Beispiel für eine Namenskonvention:	`bugfix/123456_MS_Fix_Problem_A`

Hotfix-Zweig

Der `hotfix` Branch wird verwendet, um kritische Probleme mit schwerwiegenden Auswirkungen mit minimaler Verzögerung zwischen dem Entwicklungsteam und dem in der Produktion bereitgestellten Code zu lösen. Diese Zweige sind von der `main` Filiale abgezweigt. Nachdem der Hotfix in einer Sandbox oder einer der niedrigeren Umgebungen getestet wurde, kann er auf höhere Umgebungen hochgestuft werden, indem er mit einer Merge-Anfrage zusammengeführt wird. `main` Dies ist eine vorgeschlagene Benennungskonvention für Organisation und Nachverfolgung. Dieser Prozess könnte auch mithilfe eines Feature-Branches verwaltet werden.

Benennungskonvention:	`hotfix/<ticket number>_<developer initials>_<descriptor>`
Beispiel für eine Namenskonvention:	`hotfix/123456_MS_Fix_Problem_A`

Hauptzweig

Der `main` Zweig steht immer für den Code, der in der Produktion ausgeführt wird. Code wird mithilfe von Merge-Anfragen aus `main` `feature` Verzweigungen mit dem Branch zusammengeführt. Um vor dem Löschen zu schützen und um zu verhindern, dass Entwickler Code direkt an den Branch `main` weiterleiten, aktivieren Sie den `main` Branch-Schutz.

Benennungskonvention:	`main`

Vor- und Nachteile der GitHub Flow-Strategie

Die Github Flow-Branching-Strategie eignet sich gut für kleinere, ausgereifte Entwicklungsteams mit ausgeprägten Kommunikationsfähigkeiten. Diese Strategie eignet sich gut für Teams, die Continuous Delivery implementieren möchten, und sie wird von gängigen CI/CD Engines gut unterstützt. GitHub Flow ist leichtgewichtig, hat nicht zu viele Regeln und ist in der Lage, schnelllebige Teams zu unterstützen. Es ist nicht gut geeignet, wenn Ihre Teams strenge Compliance- oder Freigabeprozesse einhalten müssen. Zusammenführungskonflikte sind in diesem Modell häufig und werden wahrscheinlich häufig auftreten. Die Lösung von Zusammenführungskonflikten ist eine Schlüsselkompetenz, und Sie müssen alle Teammitglieder entsprechend schulen.

Vorteile

GitHub Flow bietet mehrere Vorteile, die den Entwicklungsprozess verbessern, die Zusammenarbeit rationalisieren und die Gesamtqualität der Software verbessern können. Im Folgenden sind einige der wichtigsten Vorteile aufgeführt:

- Flexibel und leicht — GitHub Flow ist ein leichter und flexibler Workflow, der Entwicklern hilft, bei Softwareentwicklungsprojekten zusammenzuarbeiten. Er ermöglicht schnelle Iterationen und Experimente mit minimaler Komplexität.
- Vereinfachte Zusammenarbeit — GitHub Flow bietet einen klaren und optimierten Prozess für die Verwaltung der Funktionsentwicklung. Es fördert kleine, gezielte Änderungen, die schnell überprüft und zusammengeführt werden können, wodurch die Effizienz verbessert wird.
- Klare Versionskontrolle — Mit GitHub Flow wird jede Änderung in einem separaten Zweig vorgenommen. Dadurch wird ein klarer und nachvollziehbarer Verlauf der Versionskontrolle eingerichtet. Dies hilft Entwicklern, Änderungen nachzuverfolgen und zu verstehen, sie bei Bedarf rückgängig zu machen und eine zuverlässige Codebasis aufrechtzuerhalten.
- Nahtlose kontinuierliche Integration — GitHub Flow lässt sich in Tools für die kontinuierliche Integration integrieren. Durch die Erstellung von Pull-Requests können automatisierte Test- und Bereitstellungsprozesse eingeleitet werden. CI-Tools helfen Ihnen dabei, Änderungen gründlich zu testen, bevor sie in den `main` Branch integriert werden, wodurch das Risiko verringert wird, dass Fehler in die Codebasis aufgenommen werden.
- Schnelles Feedback und kontinuierliche Verbesserung — GitHub Flow fördert eine schnelle Feedback-Schleife, indem es häufige Codeüberprüfungen und Diskussionen durch Pull-Requests fördert. Dies erleichtert die Früherkennung von Problemen, fördert den Wissensaustausch

zwischen den Teammitgliedern und führt letztendlich zu einer höheren Codequalität und einer besseren Zusammenarbeit innerhalb des Entwicklungsteams.

- Vereinfachte Rollbacks und Rückgängigmachungen — Falls eine Codeänderung zu einem unerwarteten Fehler oder Problem führt, vereinfacht GitHub Flow den Vorgang des Rollbacks oder Rückgängigmachens der Änderung. Durch eine klare Historie von Commits und Branches ist es einfacher, problematische Änderungen zu identifizieren und rückgängig zu machen, was zur Aufrechterhaltung einer stabilen und funktionierenden Codebasis beiträgt.
- Leichte Lernkurve — GitHub Flow kann einfacher zu erlernen und anzuwenden sein als Gitflow, insbesondere für Teams, die bereits mit Git- und Versionskontrollkonzepten vertraut sind. Seine Einfachheit und sein intuitives Branching-Modell machen es für Entwickler mit unterschiedlichem Erfahrungsniveau zugänglich und reduzieren so die Lernkurve, die mit der Einführung neuer Entwicklungsworkflows verbunden ist.
- Kontinuierliche Entwicklung — GitHub Flow ermöglicht es Teams, einen kontinuierlichen Implementierungsansatz zu verfolgen, indem jede Änderung sofort implementiert wird, sobald sie in der `main` Filiale integriert ist. Dieser optimierte Prozess verhindert unnötige Verzögerungen und stellt sicher, dass die neuesten Updates und Verbesserungen den Benutzern schnell zur Verfügung gestellt werden. Dies führt zu einem agileren und reaktionsschnelleren Entwicklungszyklus.

Nachteile

GitHub Flow bietet zwar mehrere Vorteile, es ist jedoch wichtig, auch die potenziellen Nachteile zu berücksichtigen:

- Eingeschränkte Eignung für große Projekte — GitHub Flow eignet sich möglicherweise nicht so gut für Großprojekte mit komplexen Codebasen und mehreren langfristigen Funktionszweigen. In solchen Fällen könnte ein strukturierterer Workflow wie Gitflow eine bessere Kontrolle über die gleichzeitige Entwicklung und das Release-Management bieten.
- Fehlende formale Release-Struktur — GitHub Flow definiert nicht explizit einen Release-Prozess und unterstützt auch keine Funktionen wie Versionierung, Hotfixes oder Wartungszweige. Dies kann eine Einschränkung für Projekte sein, die ein striktes Release-Management erfordern oder langfristigen Support und Wartung benötigen.
- Eingeschränkte Unterstützung für langfristige Release-Planung — GitHub Flow konzentriert sich auf kurzlebige Funktionsbereiche, die möglicherweise nicht gut zu Projekten passen, die eine langfristige Release-Planung erfordern, z. B. solche mit strengen Roadmaps oder umfangreichen

Feature-Abhängigkeiten. Die Verwaltung komplexer Release-Zeitpläne kann angesichts der Einschränkungen von GitHub Flow eine Herausforderung sein.

- Potenzial für häufige Zusammenführungskonflikte — Da GitHub Flow häufiges Verzweigen und Zusammenführen fördert, besteht die Möglichkeit, dass Zusammenführungskonflikte auftreten, insbesondere bei Projekten mit viel Entwicklungsaktivität. Die Lösung dieser Konflikte kann zeitaufwändig sein und zusätzliche Anstrengungen des Entwicklungsteams erfordern.
- Fehlen formalisierter Workflow-Phasen — GitHub Flow definiert keine expliziten Entwicklungsphasen, wie Alpha-, Beta- oder Release-Candidate-Phasen. Dies kann es schwieriger machen, den aktuellen Status des Projekts oder den Stabilitätsgrad verschiedener Branches oder Releases zu kommunizieren.
- Auswirkungen bahnbrechender Änderungen — Da GitHub Flow das häufige Zusammenführen von Änderungen im `main` Branch fördert, besteht ein höheres Risiko, dass bahnbrechende Änderungen eingeführt werden, die die Stabilität der Codebasis beeinträchtigen. Strenge Codeüberprüfungs- und Testpraktiken sind entscheidend, um dieses Risiko wirksam zu mindern.

Strategie zur Branchierung von Gitflow

Gitflow ist ein Verzweigungsmodell, bei dem mehrere Zweige verwendet werden, um Code von der Entwicklung in die Produktion zu übertragen. Gitflow eignet sich gut für Teams, die Release-Zyklen geplant haben und eine Sammlung von Funktionen als Release definieren müssen.
Die Entwicklung wird in einzelnen Feature-Branches abgeschlossen, die mit Genehmigung zu einem Entwicklungszweig zusammengeführt werden, der für die Integration verwendet wird. Die Funktionen in diesem Zweig gelten als produktionsbereit. Wenn sich alle geplanten Funktionen im Entwicklungsbereich angesammelt haben, wird ein Release-Zweig für Implementierungen in höheren Umgebungen erstellt. Diese Trennung verbessert die Kontrolle darüber, welche Änderungen nach einem festgelegten Zeitplan in welche benannte Umgebung übertragen werden. Bei Bedarf können Sie diesen Prozess beschleunigen und zu einem schnelleren Bereitstellungsmodell übergehen.

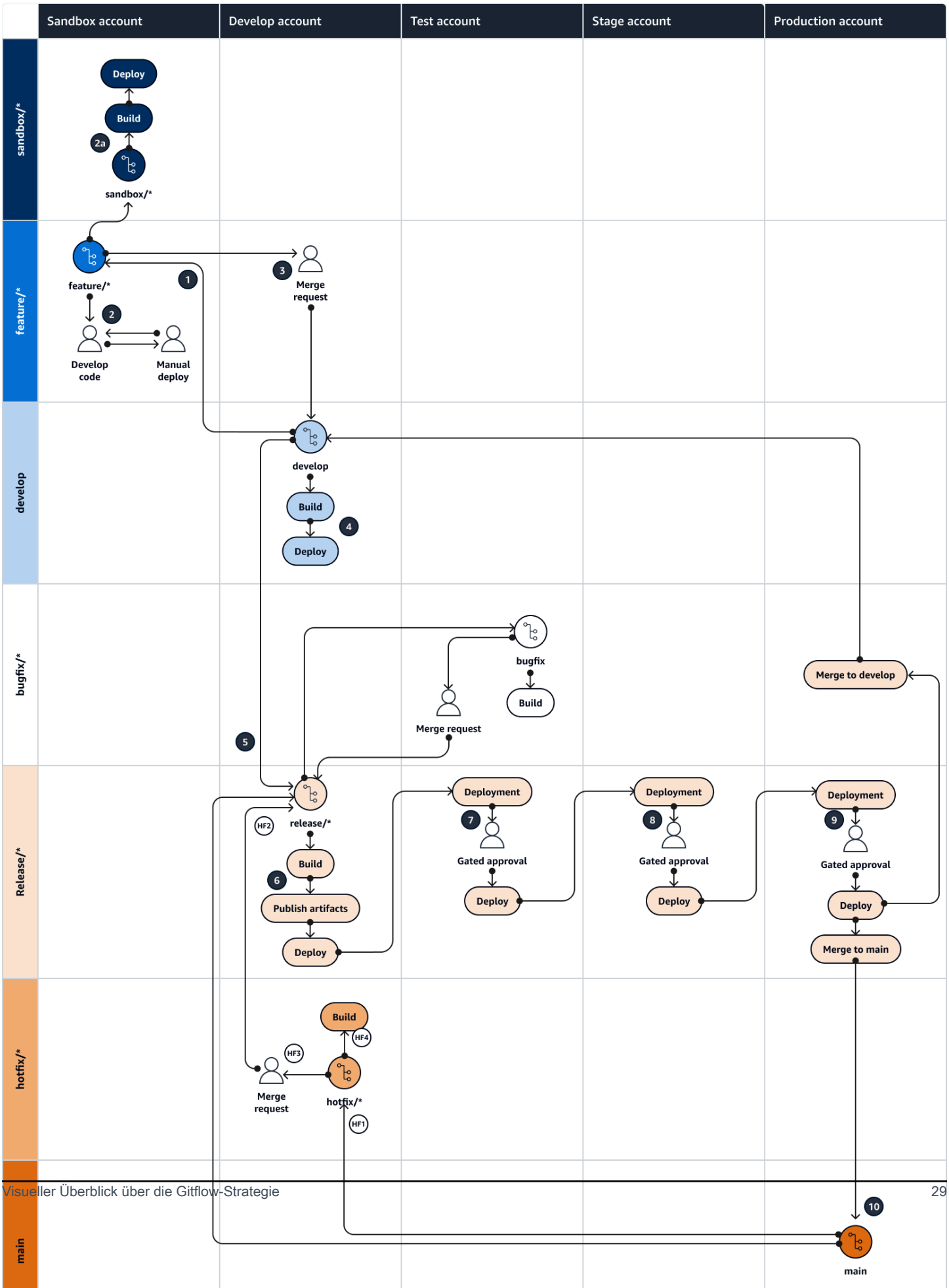
Weitere Informationen zur Gitflow-Branching-Strategie finden Sie in den folgenden Ressourcen:

- Implementieren Sie eine Gitflow-Branching-Strategie für Umgebungen mit mehreren Konten DevOps (Prescriptive Guidance)AWS
- Der ursprüngliche Gitflow-Blog (Blogbeitrag von Vincent Driessen)
- Gitflow-Workflow (Atlassian)

Themen in diesem Abschnitt:

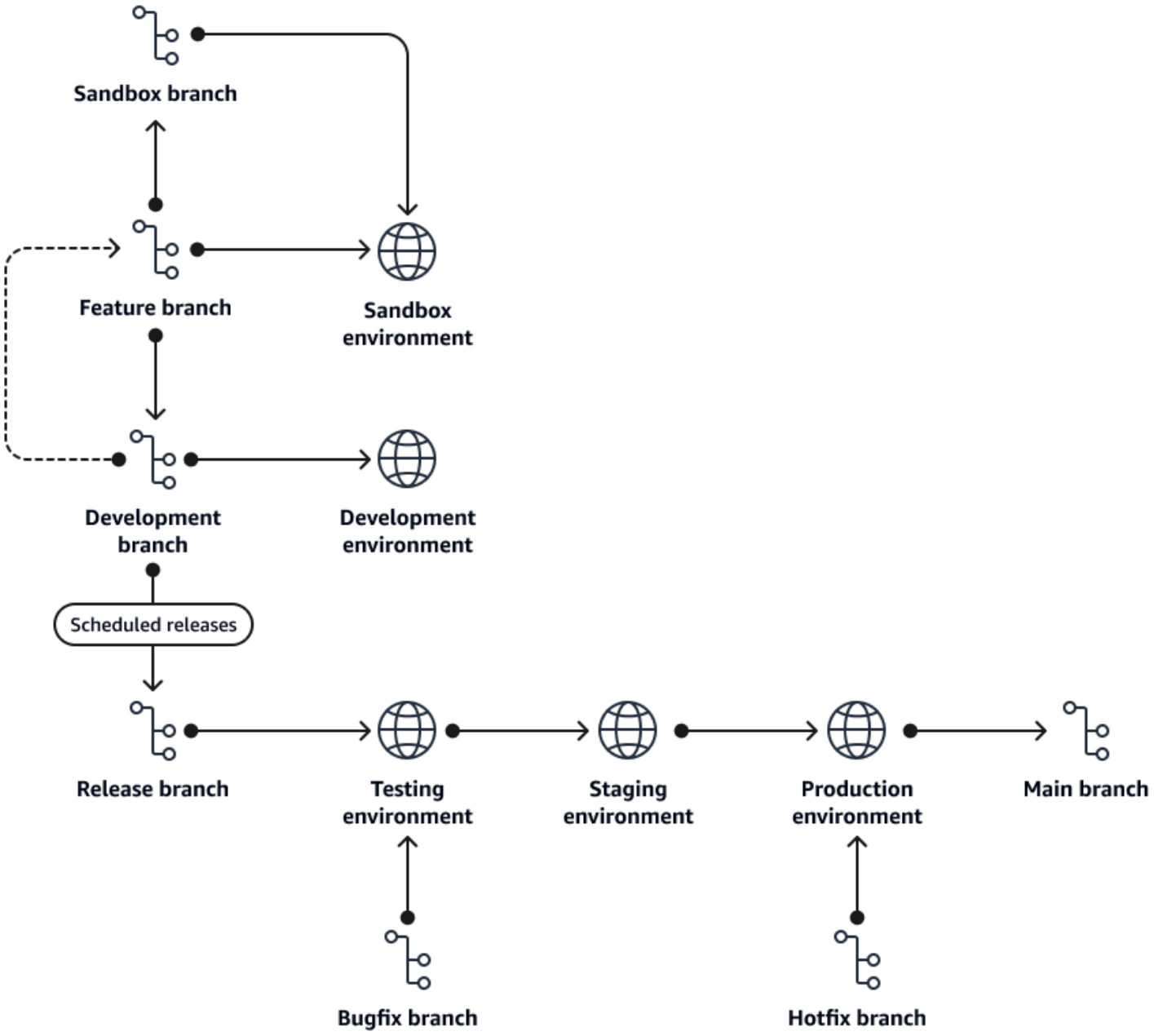
Visueller Überblick über die Gitflow-Strategie

Das folgende Diagramm kann wie ein Punnett-Quadrat verwendet werden, um die Gitflow-Verzweigungsstrategie zu verstehen. Richten Sie die Zweige auf der vertikalen Achse mit den AWS Umgebungen auf der horizontalen Achse aus, um zu bestimmen, welche Aktionen in den einzelnen Szenarien ausgeführt werden sollen. Die eingekreisten Zahlen führen Sie durch die Reihenfolge der Aktionen, die im Diagramm dargestellt sind. Weitere Informationen zu den Aktivitäten, die in den einzelnen Umgebungen stattfinden, finden Sie in diesem DevOps Handbuch unter Umgebungen.


Sandbox account
Develop account
Test account
Stage account
Production account
sandbox/*
feature/*
develop
bugfix/*
Release/*
hotfix/*
main
Deploy
Build
2a
sandbox/*
feature/*
1
2
Develop code
Manual deploy
3
Merge request
develop
Build
4
Deploy
bugfix
Build
Merge request
5
Merge to develop
HF2
release/*
Build
6
Publish artifacts
Deploy
Deployment
7
Gated approval
Deploy
Deployment
8
Gated approval
Deploy
Deployment
9
Gated approval
Deploy
Merge to main
Build
HF4
HF3
Merge request
hotfix/*
HF1
10
main

Branchen in einer Gitflow-Strategie

Eine Gitflow-Branching-Strategie umfasst üblicherweise die folgenden Zweige.



Feature-Zweig

`Feature`Branches sind kurzfristige Branches, in denen du Funktionen entwickelst. Die `feature` Verzweigung entsteht durch eine Abzweigung von der `develop` Verzweigung. Entwickler iterieren,

übertragen und testen den Code im `feature` Branch. Wenn die Funktion abgeschlossen ist, bewirbt der Entwickler die Funktion. Von einem Feature-Branch aus gibt es nur zwei Pfade vorwärts:

- Mit dem `sandbox` Zweig zusammenführen
- Erstellen Sie eine Anfrage zur Zusammenführung mit der `develop` Filiale

Benennungskonvention:	`feature/<story number>_<developer initials>_<descriptor>`
Beispiel für eine Namenskonvention:	`feature/123456_MS_Implement_Feature_A`

Sandbox-Zweig

Der `sandbox` Branch ist ein nicht standardmäßiger, kurzfristiger Branch für Gitflow. Er ist jedoch nützlich für die CI/CD Pipeline-Entwicklung. Der `sandbox` Zweig wird hauptsächlich für folgende Zwecke verwendet:

- Führen Sie eine vollständige Bereitstellung in der Sandbox-Umgebung durch, indem Sie die CI/CD Pipelines anstelle einer manuellen Bereitstellung verwenden.
- Entwickeln und testen Sie eine Pipeline, bevor Sie Zusammenführungsanfragen für vollständige Tests in einer niedrigeren Umgebung, z. B. Entwicklung oder Tests, einreichen.

`Sandbox`Zweige sind temporärer Natur und nicht für eine lange Lebensdauer konzipiert. Sie sollten nach Abschluss der spezifischen Tests gelöscht werden.

Benennungskonvention:	`sandbox/<story number>_<developer initials>_<descriptor>`
Beispiel für eine Namenskonvention:	`sandbox/123456_MS_Test_Pipeline_Deploy`

Zweig entwickeln

Der `develop` Zweig ist ein langlebiger Zweig, in dem Funktionen integriert, erstellt, validiert und in der Entwicklungsumgebung bereitgestellt werden. Alle `feature` Zweige sind in der `develop` Filiale zusammengeführt. Zusammenführungen mit der `develop` Filiale werden über eine Zusammenführungsanfrage abgeschlossen, für die ein erfolgreicher Build und zwei Genehmigungen durch Entwickler erforderlich sind. Um das Löschen zu verhindern, aktivieren Sie den Branch-Schutz für den `develop` Branch.

Benennungskonvention:	`develop`

Zweig freigeben

In Gitflow sind `release` Branches kurzfristige Branches. Diese Branches sind besonders, weil du sie in mehreren Umgebungen bereitstellen kannst, wobei du die Build-Once-Deploy-Many-Methode verwendest. `Release`Branches können auf Test-, Staging- oder Produktionsumgebungen abzielen. Nachdem ein Entwicklungsteam beschlossen hat, Funktionen in höheren Umgebungen einzuführen, erstellt es einen neuen `release` Branch und verwendet inkrementell die Versionsnummer der vorherigen Version. An den Eingängen in jeder Umgebung müssen Bereitstellungen manuell genehmigt werden, um fortzufahren. `Release`Für Branches sollte eine Merge-Anfrage erforderlich sein, damit sie geändert werden kann.

Nachdem der `release` Branch in Betrieb genommen wurde, sollte er wieder mit den `main` Zweigen `develop` und zusammengeführt werden, um sicherzustellen, dass alle Bugfixes oder Hotfixes wieder in future Entwicklungsbemühungen einfließen.

Benennungskonvention:	`release/v{major}.{minor}`
Beispiel für eine Namenskonvention:	`release/v1.0`

Hauptzweig

Der `main` Zweig ist ein langlebiger Zweig, der immer den Code darstellt, der in der Produktion ausgeführt wird. Nach einer erfolgreichen Bereitstellung aus der `main` Release-Pipeline wird Code automatisch aus einem Release-Branch mit dem Branch zusammengeführt. Um das Löschen zu verhindern, aktivieren Sie den Branch-Schutz für den `main` Branch.

Benennungskonvention:	`main`

Bugfix-Zweig

Der `bugfix` Branch ist ein kurzfristiger Branch, der verwendet wird, um Probleme in Release-Branches zu beheben, die noch nicht für die Produktion freigegeben wurden. Ein `bugfix` Branch sollte nur verwendet werden, um Fixes in `release` Branches in der Test-, Staging- oder Produktionsumgebung weiterzuleiten. Ein `bugfix` Zweig ist immer von einem `release` Zweig abgezweigt.

Nachdem der Bugfix getestet wurde, kann er durch eine Merge-Anfrage in den `release` Branch hochgestuft werden. Anschließend können Sie den `release` Branch weiterentwickeln, indem Sie dem Standard-Release-Prozess folgen.

Benennungskonvention:	`bugfix/<ticket>_<developer initials>_<descriptor>`
Beispiel für eine Namenskonvention:	`bugfix/123456_MS_Fix_Problem_A`

Hotfix-Zweig

Der `hotfix` Zweig ist ein kurzfristiger Zweig, der zur Behebung von Problemen in der Produktion verwendet wird. Sie dient ausschließlich der Förderung von Problembehebungen, die schnellstmöglich in der Produktionsumgebung eingeführt werden müssen. Ein `hotfix` Zweig wird immer abgezweigt von. `main`

Nachdem der Hotfix getestet wurde, können Sie ihn mithilfe einer Merge-Anfrage in den `release` Branch, aus dem er erstellt wurde, zur Produktion hochstufen. `main` Zu Testzwecken können Sie den `release` Branch dann weiterentwickeln, indem Sie dem Standard-Release-Prozess folgen.

Benennungskonvention:	`hotfix/<ticket>_<developer initials>_<descriptor>`
Beispiel für eine Namenskonvention:	`hotfix/123456_MS_Fix_Problem_A`

Vor- und Nachteile der Gitflow-Strategie

Die Gitflow-Branching-Strategie eignet sich gut für größere, stärker verteilte Teams, die strenge Freigabe- und Compliance-Anforderungen haben. Gitflow trägt zu einem vorhersehbaren Release-Zyklus für das Unternehmen bei, was häufig von größeren Organisationen bevorzugt wird. Gitflow eignet sich auch gut für Teams, die Leitplanken benötigen, um ihren Softwareentwicklungszyklus ordnungsgemäß abzuschließen. Dies liegt daran, dass in die Strategie mehrere Möglichkeiten zur Überprüfung und Qualitätssicherung integriert sind. Gitflow eignet sich auch gut für Teams, die mehrere Versionen von Produktionsversionen gleichzeitig verwalten müssen. Einige Nachteile von GitFlow bestehen darin, dass es komplexer ist als andere Branching-Modelle und die strikte Einhaltung des Musters erfordert, um erfolgreich abgeschlossen zu werden. Gitflow eignet sich aufgrund der starren Art der Verwaltung von Release-Branches nicht gut für Unternehmen, die eine kontinuierliche Bereitstellung anstreben. Gitflow-Release-Branches können langlebige Branches sein, in denen sich technische Schulden anhäufen können, wenn sie nicht rechtzeitig behoben werden.

Vorteile

Gitflow-based Die Entwicklung bietet mehrere Vorteile, die den Entwicklungsprozess verbessern, die Zusammenarbeit rationalisieren und die Gesamtqualität der Software verbessern können. Im Folgenden sind einige der wichtigsten Vorteile aufgeführt:

- Vorhersehbarer Release-Prozess — Gitflow folgt einem regelmäßigen und vorhersehbaren Release-Prozess. Es eignet sich gut für Teams mit regelmäßigen Entwicklungs- und Veröffentlichungszyklen.
- Verbesserte Zusammenarbeit — Gitflow fördert die Verwendung von `feature` und `release` Branches. Diese beiden Zweige helfen Teams dabei, parallel und mit minimalen Abhängigkeiten voneinander zu arbeiten.
- Gut geeignet für mehrere Umgebungen — Gitflow verwendet `release` Branches, bei denen es sich um langlebigere Branches handeln kann. Diese Branches ermöglichen es Teams, einzelne Releases über einen längeren Zeitraum hinweg ins Visier zu nehmen.
- Mehrere Versionen in der Produktion — Wenn Ihr Team mehrere Versionen der Software in der Produktion unterstützt, unterstützen `release` Gitflow-Branches diese Anforderung.
- Built-in Überprüfungen der Codequalität — Gitflow verlangt und fördert die Verwendung von Codeprüfungen und -genehmigungen, bevor Code in eine andere Umgebung übertragen wird. Dieser Prozess beseitigt Reibungspunkte zwischen Entwicklern, da dieser Schritt für alle Code-Werbeaktionen erforderlich ist.

- Vorteile für die Organisation — Gitflow hat auch auf Organisationsebene Vorteile. Gitflow empfiehlt die Verwendung eines Standard-Release-Zyklus, der der Organisation hilft, den Release-Zeitplan zu verstehen und zu antizipieren. Da das Unternehmen jetzt weiß, wann neue Funktionen bereitgestellt werden können, gibt es weniger Reibungsverluste bei den Zeitplänen, da es feste Liefertermine gibt.

Nachteile

Gitflow-based Die Entwicklung hat einige Nachteile, die sich auf den Entwicklungsprozess und die Teamdynamik auswirken können. Im Folgenden sind einige bemerkenswerte Nachteile aufgeführt:

- Komplexität — Gitflow ist ein komplexes Muster, das neue Teams erlernen müssen, und du musst dich an die Regeln von Gitflow halten, um es erfolgreich zu nutzen.
- Kontinuierliche Bereitstellung — Gitflow passt nicht zu einem Modell, bei dem viele Implementierungen schnell für die Produktion freigegeben werden. Das liegt daran, dass Gitflow die Verwendung mehrerer Branches und einen strikten Workflow für den Branch erfordert. `release`
- Filialverwaltung — Gitflow verwendet viele Branches, deren Wartung aufwändig werden kann. Es kann schwierig sein, die verschiedenen Branches nachzuverfolgen und den veröffentlichten Code zusammenzuführen, um die Branches korrekt aufeinander abzustimmen.
- Technische Schulden — Da Gitflow-Releases in der Regel langsamer sind als die anderen Branching-Modelle, können sich bis zur Veröffentlichung mehr Funktionen ansammeln, was dazu führen kann, dass sich technische Schulden anhäufen.

Teams sollten diese Nachteile sorgfältig abwägen, wenn sie entscheiden, ob Gitflow-based Entwicklung der richtige Ansatz für ihr Projekt ist.

Nächste Schritte

In diesem Leitfaden werden die Unterschiede zwischen drei gängigen Git-Branching-Strategien erklärt: GitHub Flow, Gitflow und Trunk. Es beschreibt ihre Workflows im Detail und bietet auch die Vor- und Nachteile der einzelnen Workflows. Die nächsten Schritte bestehen darin, einen dieser Standardworkflows für Ihr Unternehmen auszuwählen. Informationen zur Implementierung einer dieser Verzweigungsstrategien finden Sie im Folgenden:

- Implementieren Sie eine Trunk-Branching-Strategie für Umgebungen mit mehreren Konten DevOps
- Implementieren Sie eine GitHub Flow-Branching-Strategie für Umgebungen mit mehreren Konten DevOps
- Implementieren Sie eine Gitflow-Branching-Strategie für Umgebungen mit mehreren Konten DevOps

Wenn du dir nicht sicher bist, wo dein Team die Einführung von Git und DevOps Prozessen beginnen soll, empfehlen wir dir, eine Standardlösung auszuwählen und sie zu testen. Die Verwendung einer Standard-Branching-Konvention hilft dem Team, auf der vorhandenen Dokumentation aufzubauen und herauszufinden, was für das Team am besten funktioniert.

Haben Sie keine Angst davor, Ihre Strategie zu ändern, wenn sie für Ihr Unternehmen oder Ihre Entwicklungsteams nicht funktioniert. Die Bedürfnisse und Anforderungen von Entwicklungsteams können sich im Laufe der Zeit ändern, und es gibt keine einzige, perfekte Lösung.

Ressourcen

In diesem Leitfaden sind keine Schulungen für Git enthalten. Es stehen jedoch viele hochwertige Ressourcen im Internet zur Verfügung, falls Sie diese Schulung benötigen. Wir empfehlen, dass Sie mit der Git-Dokumentationsseite beginnen.

Die folgenden Ressourcen können Ihnen bei Ihrer Git-Branching-Reise in der AWS Cloud helfen.

AWS Präskriptive Leitlinien

- Implementieren Sie eine Trunk-Branching-Strategie für Umgebungen mit mehreren Konten DevOps
- Implementieren Sie eine GitHub Flow-Branching-Strategie für Umgebungen mit mehreren Konten DevOps
- Implementieren Sie eine Gitflow-Branching-Strategie für Umgebungen mit mehreren Konten DevOps

Weitere Hinweise AWS

- AWS DevOps Beratung
- AWS Referenzarchitektur für die Bereitstellungspipeline
- Was ist DevOps?
- DevOpsRessourcen

Sonstige Ressourcen

- Zwölf-Faktoren-App-Methodik (12factor.net)
- Geschenk-Geheimnisse (2) GitHub
- Gitflow
  - Der ursprüngliche Gitflow-Blog (Blogbeitrag von Vincent Driessen)
  - Gitflow-Workflow (Atlassian)
  - Gitflow on GitHub: So verwendest du Git Flow-Workflows mit GitHub Based Repos (Video) YouTube

  - Beispiel für Git Flow Init (YouTube Video)
  - Der Gitflow-Release-Zweig von Anfang bis Ende (Video) YouTube
- GitHub Fluss
  - GitHub Flow Quickstart (GitHub Dokumentation)
  - Warum GitHub Flow? (GitHub Flow-Webseite)
- Kofferraum
  - Einführung in die Trunk-basierte Entwicklung (Website zur Trunk-basierten Entwicklung)

Mitwirkende

Inhaltserstellung

- Mike Stephens, leitender Architekt für Cloud-Anwendungen, AWS
- Rayjan Wilson, leitender Architekt für Cloud-Anwendungen, AWS
- Abhilash Vinod, Teamleiter, leitender Architekt für Cloud-Anwendungen, AWS

Überprüfung

- Stephen DiCato, leitender Sicherheitsberater, AWS
- Gaurav Samudra, Architekt für Cloud-Anwendungen, AWS
- Steven Guggenheimer, Teamleiter, leitender Architekt für Cloud-Anwendungen, AWS

Technisches Schreiben

- Lilly AbouHarb, leitende technische Redakteurin, AWS

Dokumentverlauf

In der folgenden Tabelle werden wichtige Änderungen in diesem Leitfaden beschrieben. Um Benachrichtigungen über zukünftige Aktualisierungen zu erhalten, können Sie einen RSS-Feed abonnieren.

Änderung	Beschreibung	Datum
Aktualisieren	Wir haben die Bilder in den Abschnitten Branches in einer Trunk-Strategie, Branches in einer GitHub Flow-Strategie und Branches in einer Gitflow-Strategie ersetzt.	5. Juni 2026
Erste Veröffentlichung	—	15. Februar 2024

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.