



マルチアカウント DevOps 環境の Git 分岐戦略の選択

AWS 規範ガイドンス



AWS 規範ガイド: マルチアカウント DevOps 環境の Git 分岐戦略の選択

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon の商標およびトレードドレスは Amazon 以外の製品およびサービスに使用することはできません。また、お客様に誤解を与える可能性がある形式で、または Amazon の信用を損なう形式で使用することもできません。Amazon が所有していないその他のすべての商標は Amazon との提携、関連、支援関係の有無にかかわらず、それら該当する所有者の資産です。

Table of Contents

序章	1
目的	1
CI/CD プラクティスの使用	2
DevOps 環境について	4
サンドボックス環境	5
アクセス	5
ビルドステップ	5
デプロイ手順	5
開発環境に移行する前の期待	6
デベロッパー環境	6
アクセス	5
ビルドステップ	5
デプロイ手順	5
テスト環境に移行する前の期待	7
環境のテスト	7
アクセス	5
ビルドステップ	5
デプロイ手順	5
ステージング環境に移行する前の期待	9
ステージング環境	9
アクセス	5
ビルドステップ	5
デプロイ手順	5
本番環境に移行する前の期待	10
本番環境	10
アクセス	5
ビルドステップ	5
デプロイ手順	5
Git ベースの開発のベストプラクティス	12
Git 分岐戦略	14
トランク分岐戦略	14
トランク戦略の視覚的な概要	15
トランク戦略のブランチ	16
トランク戦略の利点と欠点	18

GitHub フロー分岐戦略	20
GitHub フロー戦略の視覚的な概要	21
GitHub フロー戦略のブランチ	22
GitHub フロー戦略の利点と欠点	24
Gitflow 分岐戦略	26
Gitflow 戦略の視覚的な概要	27
Gitflow 戦略のブランチ	29
Gitflow 戦略の利点と欠点	32
次のステップ	35
リソース	36
AWS 規範ガイド	36
その他の AWS ガイド	36
その他のリソース	36
寄稿者	38
オーサリング	38
のレビュー	38
テクニカルライティング	38
ドキュメント履歴	39
用語集	40
#	40
A	41
B	43
C	45
D	48
E	52
F	55
G	56
H	57
I	59
L	61
M	62
O	66
P	69
Q	72
R	72
S	75

T	79
U	80
V	81
W	81
Z	82
.....	lxxxiii

マルチアカウント DevOps 環境の Git 分岐戦略の選択

Amazon Web Services ([寄稿者](#))

2024 年 2 月 ([ドキュメント履歴](#))

クラウドベースのアプローチに移行し、でソフトウェアソリューションを提供することは、変革をもたらす AWS 可能性があります。これには、ソフトウェア開発ライフサイクルプロセスの変更が必要になる場合があります。通常、の開発プロセスでは複数の AWS アカウントが使用されます AWS クラウド。DevOps プロセスとペアリングする互換性のある Git 分岐戦略を選択することは、成功するために不可欠です。組織に適した Git ブランチ戦略を選択すると、開発チーム全体で DevOps 標準とベストプラクティスを簡潔に伝達できます。Git 分岐は 1 つの環境で簡単にできますが、サンドボックス、開発、テスト、ステージング、本番環境など、複数の環境に適用すると混乱する可能性があります。複数の環境があると、DevOps 実装の複雑さが増します。

このガイドでは、組織がマルチアカウント DevOps プロセスを実装する方法を示す Git 分岐戦略のビジュアル図を提供します。ビジュアルガイドは、チームが Git 分岐戦略を DevOps プラクティスとマージする方法を理解するのに役立ちます。Gitflow、GitHub Flow、Trunk などの標準分岐モデルを使用してソースコードリポジトリを管理すると、開発チームが作業を調整するのに役立ちます。これらのチームは、インターネット上の標準の Git トレーニングリソースを使用して、これらのモデルと戦略を理解して実装することもできます。

DevOps の DevOps ベストプラクティスについては AWS、AWS 「Well-Architected」の[DevOps ガイド](#)を参照してください。このガイドを確認するときは、デューデリジェンスを使用して、組織に適した分岐戦略を選択します。一部の戦略は、他の戦略よりもユースケースに適している場合があります。

目的

このガイドは、複数のを持つ組織向けの DevOps ブランチ戦略の選択と実装に関するドキュメントシリーズの一部です AWS アカウント。このシリーズは、最初から要件、目標、ベストプラクティスに最適な戦略を適用して、でのエクスペリエンスを合理化するのに役立つように設計されています AWS クラウド。このガイドには、組織が使用する継続的インテグレーションおよび継続的デリバリー (CI/CD) エンジンとテクノロジーフレームワークによって異なるため、DevOps 実行可能スクリプトは含まれていません。

このガイドでは、GitHub Flow、Gitflow、Trunk の 3 つの一般的な Git 分岐戦略の違いについて説明します。このガイドの推奨事項は、チームが組織の目標に沿った分岐戦略を特定するのに役立ちま

す。このガイドを読むと、組織の分岐戦略を選択できるようになります。戦略を選択したら、次のいずれかのパターンを使用して、開発チームでその戦略を実装できます。

- [マルチアカウント DevOps 環境のトランク分岐戦略を実装する](#)
- [マルチアカウント DevOps 環境に GitHub フロー分岐戦略を実装する](#)
- [マルチアカウント DevOps 環境に Gitflow 分岐戦略を実装する](#)

ある組織、チーム、またはプロジェクトに何が適しているかは、他の組織、チーム、またはプロジェクトには適していない可能性があることに注意してください。Git 分岐戦略の選択は、チームサイズ、プロジェクト要件、コラボレーション、統合頻度、リリース管理の望ましいバランスなど、さまざまな要因によって異なります。

CI/CD プラクティスの使用

AWS では、ソフトウェアリリースライフサイクルを自動化するプロセスである継続的インテグレーションと継続的デリバリー (CI/CD) を実装することをお勧めします。これにより、開発から本番環境に新しいコードを取得するために従来必要とされる手動 DevOps プロセスの多くまたはすべてが自動化されます。CI/CD パイプラインには、サンドボックス、開発、テスト、ステージング、および本番環境が含まれます。各環境では、CI/CD パイプラインはコードをデプロイまたはテストするために必要なインフラストラクチャをプロビジョニングします。CI/CD を使用することで、開発チームはコードを変更し、自動的にテストしてデプロイできます。CI/CD パイプラインは、開発チームのガバナンスとガードレールも提供します。機能の受け入れとデプロイには、一貫性、標準、ベストプラクティス、最小承認レベルが適用されます。詳細については、[「継続的インテグレーションと継続的デリバリーの実践 AWS」](#)を参照してください。

このガイドで説明されているすべての分岐戦略は、CI/CD プラクティスに適しています。CI/CD パイプラインの複雑さは、分岐戦略の複雑さとともに増加します。例えば、Gitflow は、このガイドで説明されている最も複雑な分岐戦略です。この戦略の CI/CD パイプラインには、より多くのステップ (コンプライアンス上の理由など) が必要であり、複数の同時本番リリースをサポートする必要があります。CI/CD の使用は、分岐戦略の複雑さが増すにつれて、より重要になります。これは、CI/CD が開発チームのガードレールとメカニズムを確立し、開発者が定義されたプロセスを意図的または意図せずに回避できないためです。

AWS は、CI/CD パイプラインの構築を支援するように設計された一連の開発者サービスを提供します。例えば、[AWS CodePipeline](#)は完全マネージド型の継続的デリバリーサービスであり、リリースパイプラインを自動化してアプリケーションとインフラストラクチャを迅速かつ確実に更新できま

す。はソースコードを[AWS CodeBuild](#)コンパイルし、テストを実行し、ready-to-deployソフトウェアパッケージを生成します。詳細については、「[Developer Tools on AWS](#)」を参照してください。

DevOps 環境について

分岐戦略を理解するには、各環境で発生する目的とアクティビティを理解する必要があります。複数の環境を確立することで、開発アクティビティをステージに分割し、それらのアクティビティをモニタリングし、未承認機能の意図しないリリースを防ぐことができます。AWS アカウント 各環境に 1 つ以上の を持つことができます。

ほとんどの組織には、いくつかの使用環境が概説されています。ただし、環境の数は組織やソフトウェア開発ポリシーによって異なる場合があります。このドキュメントシリーズでは、開発パイプラインにまたがる次の 5 つの一般的な環境があることを前提としていますが、異なる名前呼び出される場合があります。

- サンドボックス – デベロッパーがコードを記述し、間違いを犯し、概念実証作業を実行する環境。
- 開発 – 開発者がコードを統合して、すべてが単一のまとまりのあるアプリケーションとして動作することを確認する環境。
- テスト – QA チームまたは承認テストが行われる環境。多くの場合、チームはこの環境でパフォーマンステストや統合テストを行います。
- ステージング – 本番稼働前の環境で、本番稼働と同等の状況でコードとインフラストラクチャが期待どおりに動作することを検証します。この環境は、本番環境とできるだけ類似するように設定されています。
- 本番稼働 – エンドユーザーと顧客からのトラフィックを処理する環境。

このセクションでは、各環境について詳しく説明します。また、各環境のビルドステップ、デプロイステップ、終了基準についても説明し、次の段階に進むことができます。次の図は、これらの環境を順番に示しています。



このセクションのトピック:

- [サンドボックス環境](#)
- [デベロッパー環境](#)

- [環境のテスト](#)
- [ステージング環境](#)
- [本番環境](#)

サンドボックス環境

サンドボックス環境は、デベロッパーがコードを記述し、間違いを犯し、概念実証作業を実行する場所です。サンドボックス環境にデプロイするには、ローカルワークステーションから、またはローカルワークステーションのスクリプトを使用します。

アクセス

開発者はサンドボックス環境へのフルアクセスを持っている必要があります。

ビルドステップ

開発者は、サンドボックス環境に変更をデプロイする準備ができたなら、ローカルワークステーションでビルドを手動で実行します。

1. [git-secrets](#) (GitHub) を使用して機密情報をスキャンする
2. ソースコードをリントする
3. 該当する場合は、ソースコードを構築してコンパイルします。
4. ユニットテストを実行する
5. コードカバレッジ分析を実行する
6. 静的コード分析を行う
7. Infrastructure as Code (IaC) を構築する
8. IaC セキュリティ分析を実行する
9. オープンソースライセンスを抽出する
10. ビルドアーティファクトを発行する

デプロイ手順

Gitflow モデルまたは Trunk モデルを使用している場合、feature ブランチがサンドボックス環境で正常に構築されると、デプロイステップが自動的に開始されます。GitHub Flow モデルを使用してい

場合は、以下のデプロイステップを手動で実行します。サンドボックス環境のデプロイ手順は次のとおりです。

1. 公開されたアーティファクトをダウンロードする
2. データベースのバージョンングを実行する
3. IaC デプロイを実行する
4. 統合テストを実行する

開発環境に移行する前の期待

- サンドボックス環境でfeatureブランチを正常に構築する
- デベロッパーがサンドボックス環境でこの機能を手動でデプロイしてテストした

デベロッパー環境

開発環境では、デベロッパーがコードを統合して、すべてを1つのまとまりのあるアプリケーションとして動作させます。Gitflow では、開発環境にはマージリクエストに含まれる最新の機能が含まれており、リリースの準備が整います。GitHub フローおよびトランク戦略では、開発環境はテスト環境と見なされ、コードベースは不安定で本番環境へのデプロイに適していない可能性があります。

アクセス

最小特権の原則に従ってアクセス許可を割り当てます。最小特権は、タスクの実行に必要な最小限のアクセス許可を付与するというセキュリティのベストプラクティスです。開発者は、サンドボックス環境よりも開発環境へのアクセスが少なくなければなりません。

ビルドステップ

develop ブランチ (Gitflow) またはmainブランチ (Trunk または GitHub Flow) へのマージリクエストを作成すると、ビルドが自動的に開始されます。

1. [git-secrets](#) (GitHub) を使用して機密情報をスキャンする
2. ソースコードをリントする
3. 該当する場合は、ソースコードを構築してコンパイルします。
4. ユニットテストを実行する

5. コードカバレッジ分析を実行する
6. 静的コード分析を行う
7. ビルド IaC
8. IaC セキュリティ分析を実行する
9. オープンソースライセンスを抽出する

デプロイ手順

Gitflow モデルを使用している場合、develop ブランチが開発環境で正常に構築されると、デプロイステップが自動的に開始されます。GitHub フローモデルまたはトランクモデルを使用している場合、main ブランチに対してマージリクエストが作成されると、デプロイステップが自動的に開始されます。開発環境のデプロイ手順は次のとおりです。

1. ビルドステップから公開されたアーティファクトをダウンロードする
2. データベースのバージョンングを実行する
3. IaC デプロイを実行する
4. 統合テストを実行する

テスト環境に移行する前の期待

- 開発環境における develop ブランチ (Gitflow) または main ブランチ (Trunk または GitHub Flow) のビルドとデプロイの成功
- ユニットテストが 100% で合格
- IaC ビルドの成功
- デプロイアーティファクトが正常に作成されました
- 開発者が手動検証を実行して、機能が期待どおりに機能していることを確認します。

環境のテスト

品質保証 (QA) 担当者は、テスト環境を使用して機能を検証します。テストが終了した後、変更を承認します。承認されると、ブランチは次の環境、ステージングに移行します。Gitflow では、この環境とその上の環境は release ブランチからのデプロイでのみ使用できます。release ブランチは、計画された機能を含む develop ブランチに基づいています。

アクセス

最小特権の原則に従ってアクセス許可を割り当てます。開発者は、開発環境よりもテスト環境へのアクセスが少なくなければなりません。QA 担当者には、機能をテストするための十分なアクセス許可が必要です。

ビルドステップ

この環境のビルドプロセスは、Gitflow 戦略を使用する場合のバグ修正にのみ適用されます。bugfix ブランチへのマージリクエストを作成すると、ビルドが自動的に開始されます。

1. [git-secrets](#) (GitHub) を使用して機密情報をスキャンする
2. ソースコードをリントする
3. 該当する場合は、ソースコードを構築してコンパイルします。
4. ユニットテストを実行する
5. コードカバレッジ分析を実行する
6. 静的コード分析を行う
7. ビルド IaC
8. IaC セキュリティ分析を実行する
9. オープンソースライセンスを抽出する

デプロイ手順

開発環境にデプロイした後、テスト環境で release ブランチ (Gitflow) または main ブランチ (Trunk または GitHub Flow) のデプロイを自動的に開始します。テスト環境でのデプロイ手順は次のとおりです。

1. release ブランチ (Gitflow) または main ブランチ (Trunk または GitHub Flow) をテスト環境にデプロイする
2. 指定された担当者による手動承認のために一時停止する
3. 公開されたアーティファクトをダウンロードする
4. データベースのバージョンングを実行する
5. IaC デプロイを実行する
6. 統合テストを実行する
7. パフォーマンステストを実行する

8. 品質保証の承認

ステージング環境に移行する前の期待

- 開発チームと QA チームは、組織の要件を満たすのに十分なテストを実施しました。
- 開発チームは、bugfix ブランチを通じて検出されたバグを解決しました。

ステージング環境

ステージング環境は、本番環境と同じになるように設定されています。たとえば、データ設定の範囲とサイズは本番ワークロードと類似している必要があります。ステージング環境を使用して、コードとインフラストラクチャが期待どおりに動作することを確認します。この環境は、プレビューや顧客デモなどのビジネスユースケースにも推奨されます。

アクセス

最小特権の原則に従ってアクセス許可を割り当てます。開発者は、本番環境と同じステージング環境にアクセスできる必要があります。

ビルドステップ

なし。テスト環境で使用されたものと同じアーティファクトがステージング環境で再利用されます。

デプロイ手順

テスト環境での承認とデプロイ後、ステージング環境で release ブランチ (Gitflow) または main ブランチ (Trunk または GitHub Flow) のデプロイを自動的に開始します。ステージング環境でのデプロイ手順は次のとおりです。

1. release ブランチ (Gitflow) または main ブランチ (Trunk または GitHub Flow) をステージング環境にデプロイする
2. 指定された担当者による手動承認のために一時停止する
3. 公開されたアーティファクトをダウンロードする
4. データベースのバージョンングを実行する
5. IaC デプロイを実行する
6. (オプション) 統合テストを実行する

7. (オプション) 負荷テストを実行する
8. 必要な開発、QA、製品、またはビジネス承認者から承認を得る

本番環境に移行する前の期待

- 本番環境と同等のリリースがステージング環境に正常にデプロイされました
- (オプション) 統合と負荷テストが成功しました

本番環境

実稼働環境は、リリースされた製品をサポートし、実際のクライアントによって実際のデータを処理します。これは、最小特権でアクセスが割り当てられた保護された環境であり、昇格されたアクセスは、監査された例外プロセスを通じてのみ一定期間許可する必要があります。

アクセス

本番環境では、デベロッパーは AWS マネジメントコンソールでの読み取り専用アクセスが制限されている必要があります。たとえば、デベロッパーは day-to-day のログデータにアクセスできる必要があります。本番環境へのすべてのリリースは、デプロイ前に承認ステップでゲートする必要があります。

ビルドステップ

なし。テスト環境とステージング環境で使用されたものと同じアーティファクトが本番環境で再利用されます。

デプロイ手順

承認後、本番環境で release ブランチ (Gitflow) または main ブランチ (Trunk または GitHub Flow) のデプロイを自動的に開始し、ステージング環境にデプロイします。以下は、本番環境でのデプロイ手順です。

1. release ブランチ (Gitflow) または main ブランチ (Trunk または GitHub Flow) を本番環境にデプロイする
2. 指定された担当者による手動承認のために一時停止する
3. 公開されたアーティファクトをダウンロードする

4. データベースのバージョンングを実行する
5. IaC デプロイを実行する

Git ベースの開発のベストプラクティス

Git ベースの開発を正常に採用するには、コラボレーションを促進し、コード品質を維持し、継続的インテグレーションと継続的デリバリー (CI/CD) をサポートする一連のベストプラクティスに従うことが重要です。このガイドのベストプラクティスに加えて、[AWS Well-Architected DevOps ガイダンス](#)を参照してください。以下は、Git ベースの開発の主なベストプラクティスです AWS。

- 小規模かつ頻繁な変更を維持する — 開発者が小規模で段階的な変更や機能をコミットすることを奨励します。これにより、マージ競合のリスクが軽減され、問題をすばやく特定して修正することが容易になります。
- 機能トグルの使用 – 不完全または実験的な機能のリリースを管理するには、機能トグルまたは機能フラグを使用します。これにより、メインブランチの安定性に影響を与えずに、本番環境の特定の機能を非表示、有効化、または無効化できます。
- 堅牢なテストスイートを維持する – 問題を早期に検出し、コードベースが安定していることを検証するには、包括的で適切に保守されたテストスイートが不可欠です。テストの自動化に投資し、失敗したテストの修正に優先順位を付けます。
- 継続的インテグレーションの採用 – 継続的インテグレーションツールとプラクティスを使用して、コード変更を自動的に構築、テストし、develop ブランチ (Gitflow) または main ブランチ (Trunk または GitHub Flow) に統合します。これにより、問題を早期に把握し、開発プロセスを合理化できます。
- コードレビューの実行 – コードのピアレビューを奨励して、品質を維持し、知識を共有し、main ブランチに統合される前に潜在的な問題を見つけます。プルリクエストやその他のコードレビューツールを使用して、このプロセスを容易にします。
- 壊れたビルドを監視して修正する – ビルドが壊れたりテストに失敗した場合は、できるだけ早く問題を修正することを優先します。これにより、develop ブランチ (Gitflow) または main ブランチ (Trunk または GitHub Flow) が解放可能な状態を維持し、他のデベロッパーへの影響を最小限に抑えます。
- コミュニケーションとコラボレーション — チームメンバー間のオープンなコミュニケーションとコラボレーションを促進します。デベロッパーが進行中の作業とコードベースに加えられた変更を認識していることを確認します。
- 継続的なリファクタリング — コードベースを定期的なリファクタリングして、メンテナンス性を向上させ、技術的負債を減らします。デベロッパーに、コードを見つけたよりも良い状態のままにするよう促します。
- 複雑なタスクに存続期間の短いブランチを使用する – 大規模または複雑なタスクの場合は、存続期間の短いブランチ (タスクブランチとも呼ばれます) を使用して変更を処理します。ただし、ブ

ランチの有効期間は短く、通常は 1 日未満にしてください。できるだけ早く変更を develop ブランチ (Gitflow) または main ブランチ (Trunk または GitHub Flow) にマージします。チームが 1 つの大きなマージリクエストを使用するよりも、小規模で頻繁なマージとレビューを使用する方が簡単です。

- チームのトレーニングとサポート – Git ベースの開発を初めて使用する場合や、ベストプラクティスを採用する際にガイダンスを必要とする開発者にトレーニングとサポートを提供します。

Git 分岐戦略

このガイドでは、最小から最も複雑な順に、以下の Git ベースの分岐戦略について詳しく説明します。

- **トランク – トランクベースの開発**は、すべてのデベロッパーが単一のブランチで作業するソフトウェア開発プラクティスであり、通常は trunk または main ブランチと呼ばれます。このアプローチの背後にある考え方は、コードの変更を頻繁に統合し、自動テストと継続的な統合に依存することで、コードベースを継続的に解放可能な状態に維持することです。
- **GitHub Flow** – GitHub Flow は、GitHub によって開発された軽量のブランチベースのワークフローです。これは、存続期間の短い feature ブランチの概念に基づいています。機能が完了し、デプロイする準備ができると、その機能は main ブランチにマージされます。
- **Gitflow** – Gitflow アプローチでは、開発は個々の機能ブランチで完了します。承認後、feature ブランチを、通常は という名前の統合ブランチにマージします develop。develop ブランチに十分な機能が蓄積されると、その機能を上位環境にデプロイするための release ブランチが作成されます。

各分岐戦略には利点と欠点があります。すべて同じ環境を使用しますが、すべて同じブランチや手動の承認ステップを使用するわけではありません。ガイドのこのセクションでは、各分岐戦略の詳細を確認して、そのニュアンスを理解し、組織のユースケースに適しているかどうかを評価できるようにします。

このセクションのトピック:

- [トランク分岐戦略](#)
- [GitHub フロー分岐戦略](#)
- [Gitflow 分岐戦略](#)

トランク分岐戦略

トランクベースの開発は、すべてのデベロッパーが単一のブランチで作業するソフトウェア開発プラクティスであり、通常は trunk または main ブランチと呼ばれます。このアプローチの背後にある考え方は、コードの変更を頻繁に統合し、自動テストと継続的な統合に依存することで、コードベースを継続的に解放可能な状態に維持することです。

トランクベースの開発では、開発者は 1 日に複数回 main ブランチに変更をコミットし、小規模で増分的な更新を目指します。これにより、迅速なフィードバックループが可能になり、マージ競合のリスクが軽減され、チームメンバー間のコラボレーションが促進されます。このプラクティスでは、メンテナンスが適切に行われているテストスイートの重要性を強調しています。これは、潜在的な問題を早期に検出し、コードベースが安定し、解放可能であることを確認するために、自動テストに依存しているためです。

トランクベースの開発は、多くの場合、機能ベースの開発 (機能ブランチまたは機能駆動型開発とも呼ばれます) とは対照的です。各新機能またはバグ修正は、メインブランチとは別の独自の専用ブランチで開発されます。トランクベースの開発と機能ベースの開発のどちらを選択するかは、チームサイズ、プロジェクト要件、コラボレーション、統合頻度、リリース管理の望ましいバランスなどの要因によって異なります。

トランク分岐戦略の詳細については、次のリソースを参照してください。

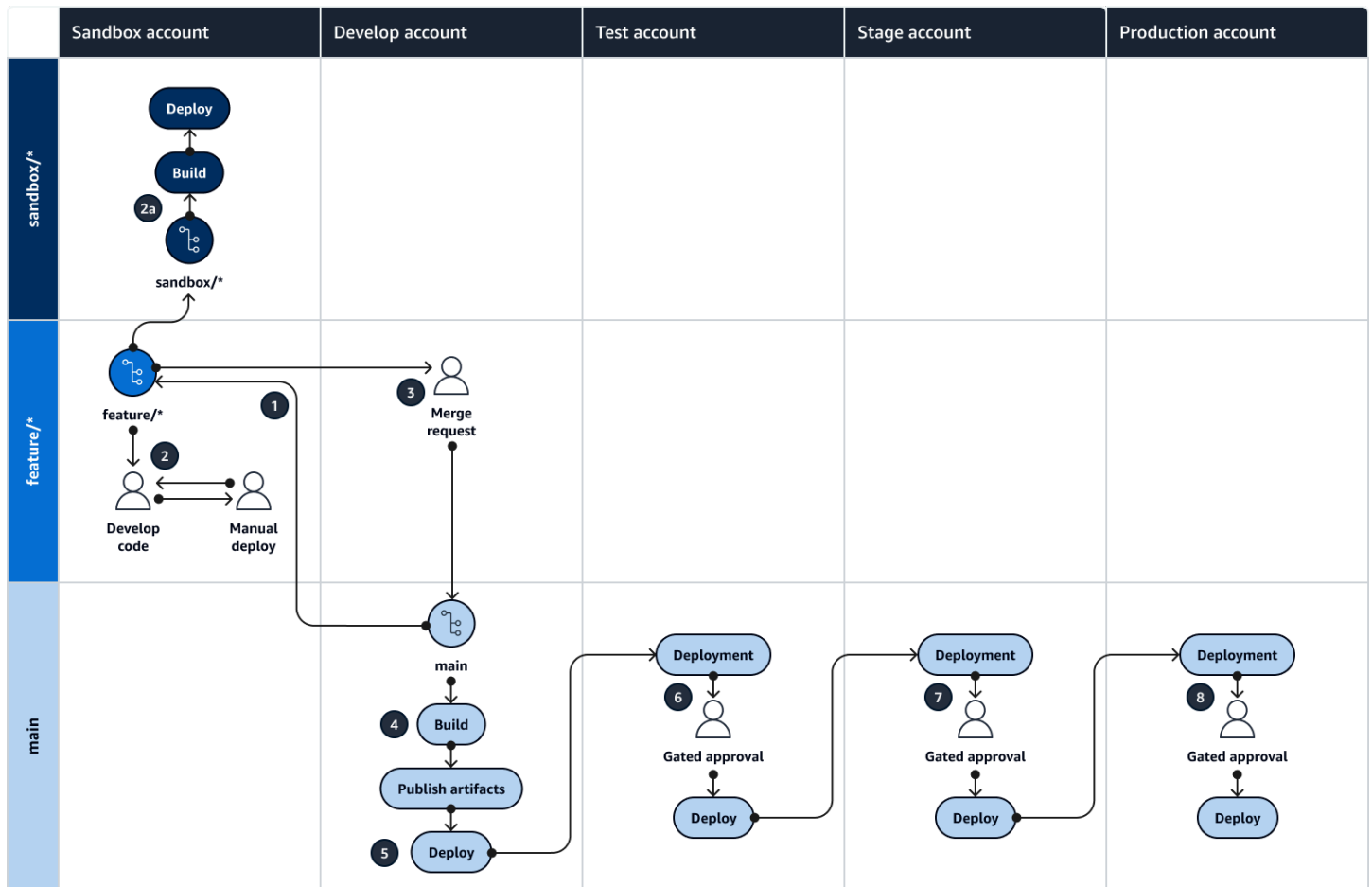
- [マルチアカウント DevOps 環境のトランク分岐戦略を実装する \(AWS 規範ガイド\)](#)
- [トランクベースの開発の概要 \(トランクベースの開発ウェブサイト\)](#)

このセクションのトピック:

- [トランク戦略の視覚的な概要](#)
- [トランク戦略のブランチ](#)
- [トランク戦略の利点と欠点](#)

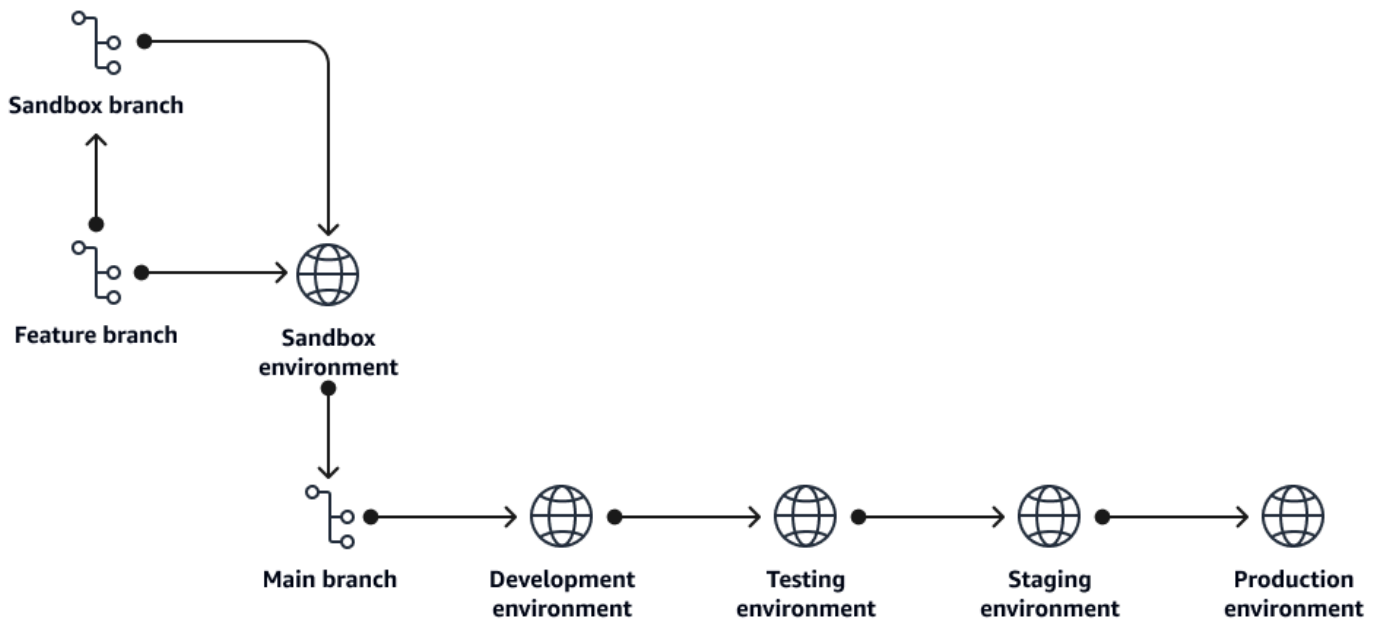
トランク戦略の視覚的な概要

次の図は、トランク分岐戦略を理解するために、[Punnett 二乗](#) (Wikipedia) のように使用できます。垂直軸のブランチを水平軸の AWS 環境と並べて、各シナリオで実行するアクションを決定します。円で囲まれた数字は、図に示されている一連のアクションをガイドします。この図は、サンドボックス環境のブランチから feature ブランチの本番リリースまで、トランクブランチ戦略の開発ワークフローを示しています main。各環境で発生するアクティビティの詳細については、このガイドの [DevOps 環境](#)」を参照してください。



トランク戦略のブランチ

トランク分岐戦略には通常、次のブランチがあります。



機能ブランチ

feature ブランチで機能を開発するか、ホットフィックスを作成します。feature ブランチを作成するには、main ブランチから分岐します。開発者は、feature ブランチでコードを反復、コミット、テストします。機能が完了すると、開発者はその機能を昇格させます。feature ブランチから転送されるパスは 2 つだけです。

- sandbox ブランチにマージする
- main ブランチへのマージリクエストを作成する

命名規則: `feature/<story number>_<developer initials>_<descriptor>`

命名規則の例: `feature/123456_MS_Implement _Feature_A`

サンドボックスブランチ

このブランチは非標準トランクブランチですが、CI/CD パイプライン開発に役立ちます。sandbox ブランチは主に以下の目的に使用されます。

- CI/CD パイプラインを使用してサンドボックス環境への完全なデプロイを実行する

- 開発やテストなど、より低い環境で完全なテストのためのマージリクエストを送信する前に、パイプラインを開発してテストします。

Sandbox ブランチは一時的なものであり、短期間の使用を想定しています。これらは、特定のテストが完了した後に削除する必要があります。

命名規則: `sandbox/<story number>_<developer initials>_<descriptor>`

命名規則の例: `sandbox/123456_MS_Test_Pipeline_Deploy`

メインブランチ

main ブランチは常に、本番環境で実行されているコードを表します。コードは から分岐され main、開発され、 にマージされます main。からのデプロイは、任意の環境をターゲットに main できます。削除から保護するには、ブランチの main ブランチ保護を有効にします。

命名規則: `main`

修正ブランチ

トランクベースのワークフローには専用 hotfix ブランチはありません。ホットフィックスは feature ブランチを使用します。

トランク戦略の利点と欠点

トランク分岐戦略は、強力なコミュニケーションスキルを持つ、成熟した小規模な開発チームに適しています。また、アプリケーションの継続的なローリング機能リリースがある場合にも有効です。大規模または断片化された開発チームがある場合や、機能リリースが大規模でスケジュールされている場合には適していません。このモデルではマージ競合が発生するため、マージ競合の解決が重要なスキルであることに注意してください。すべてのチームメンバーは、それに応じてトレーニングを受ける必要があります。

利点

トランクベースの開発には、開発プロセスを改善し、コラボレーションを合理化し、ソフトウェアの全体的な品質を向上させることができるいくつかの利点があります。以下は、主な利点の一部です。

- フィードバックループの高速化 – トランクベースの開発により、デベロッパーはコードの変更を頻繁に統合し、多くの場合は 1 日に複数回統合します。これにより、潜在的な問題に関するフィードバックが速くなり、開発者は機能ベースの開発モデルよりも迅速に問題を特定して修正できます。
- マージ競合の軽減 – トランクベースの開発では、変更が継続的に統合されるため、大規模で複雑なマージ競合のリスクが最小限に抑えられます。これにより、よりクリーンなコードベースを維持し、競合の解決に費やす時間を短縮できます。機能ベースの開発では、競合の解決に時間がかかり、エラーが発生しやすくなります。
- コラボレーションの向上 – トランクベースの開発により、開発者は同じブランチで協力し、チーム内でのコミュニケーションとコラボレーションが向上します。これにより、問題解決が迅速になり、チームダイナミクスが強化される可能性があります。
- コードレビューの簡素化 – コードの変更はトランクベースの開発ではより小さく、より頻繁に行われるため、詳細なコードレビューを簡単に実行できます。小さな変更では、一般的に理解とレビューが容易になり、潜在的な問題と改善点をより効果的に特定できます。
- 継続的インテグレーションとデリバリー – トランクベースの開発は、継続的インテグレーションと継続的デリバリー (CI/CD) の原則をサポートしています。コードベースを解放可能な状態に保ち、変更を頻繁に統合することで、チームは CI/CD プラクティスをより簡単に採用できるため、デプロイサイクルが短縮され、ソフトウェア品質が向上します。
- コード品質の向上 – 頻繁な統合、テスト、コードレビューにより、トランクベースの開発は全体的なコード品質の向上に貢献できます。開発者は問題をより迅速に把握して修正できるため、時間の経過とともに技術的負債が蓄積される可能性が低くなります。
- ブランチ戦略の簡素化 – トランクベースの開発では、存続期間の長いブランチの数を減らすことで、ブランチ戦略を簡素化します。これにより、特に大規模なプロジェクトやチームでは、コードベースの管理と保守が容易になります。

欠点

トランクベースの開発にはいくつかの欠点があり、開発プロセスとチームのダイナミクスに影響を与える可能性があります。以下は、いくつかの注目すべき欠点です。

- 制限された分離 – すべてのデベロッパーが同じブランチで作業するため、その変更はチームの全員にすぐに表示されます。これにより、干渉や競合が発生し、意図しない副作用が発生したり、ビルドが壊れたりする可能性があります。これとは対照的に、機能ベースの開発では変更をより適切に分離できるため、開発者はより独立して作業できます。
- テストへのプレッシャーの高まり – トランクベースの開発では、問題を迅速に検出するために、継続的な統合と自動テストに依存しています。ただし、このアプローチはテストインフラストラクチャに大きな負荷をかける可能性があります。適切に保守されたテストスイートが必要です。テストが包括的でも信頼性でもない場合、メインブランチで検出されない問題が発生する可能性があります。
- リリースに対する制御の軽減 – トランクベースの開発は、コードベースを継続的に解放可能な状態に維持することを目指しています。これは有利ですが、リリーススケジュールが厳しいプロジェクトや、特定の機能を一緒にリリースする必要があるプロジェクトに必ずしも適しているとは限りません。機能ベースの開発では、機能がいつどのようにリリースされるかをより細かく制御できます。
- コードチャーン – 開発者が常に変更をメインブランチに統合しているため、トランクベースの開発はコードチャーンの増加につながる可能性があります。これにより、開発者がコードベースの現在の状態を追跡することが困難になり、最近の変更の影響を理解しようとすると混乱が生じる可能性があります。
- 強力なチーム文化が必要 – トランクベースの開発には、チームメンバー間の高レベルの規律、コミュニケーション、コラボレーションが必要です。これは、特に大規模なチームや、このアプローチの経験の浅い開発者と作業する場合に、維持するのが難しい場合があります。
- スケーラビリティの課題 – 開発チームの規模が大きくなるにつれて、メインブランチに統合されるコード変更の数は急速に増加する可能性があります。これにより、ビルドブレイクやテスト障害が頻繁に発生し、コードベースを解放可能な状態に維持することが困難になる可能性があります。

GitHub フロー分岐戦略

GitHub Flow は、GitHub によって開発された軽量のブランチベースのワークフローです。GitHub Flow は、機能が完了し、デプロイの準備が整ったときにメインブランチにマージされる、存続期間の短い機能ブランチの概念に基づいています。GitHub Flow の主な原則は次のとおりです。

- ブランチ化は軽量 – 開発者は数回クリックするだけで作業用の機能ブランチを作成できるため、メインブランチに影響を与えずに共同作業や実験を行うことができます。
- 継続的デプロイ – 変更はメインブランチにマージされるとすぐにデプロイされるため、迅速なフィードバックと反復が可能になります。

- リクエストのマージ – 開発者はマージリクエストを使用してディスカッションを開始し、変更をメインブランチにマージする前にプロセスを確認します。

GitHub Flow の詳細については、次のリソースを参照してください。

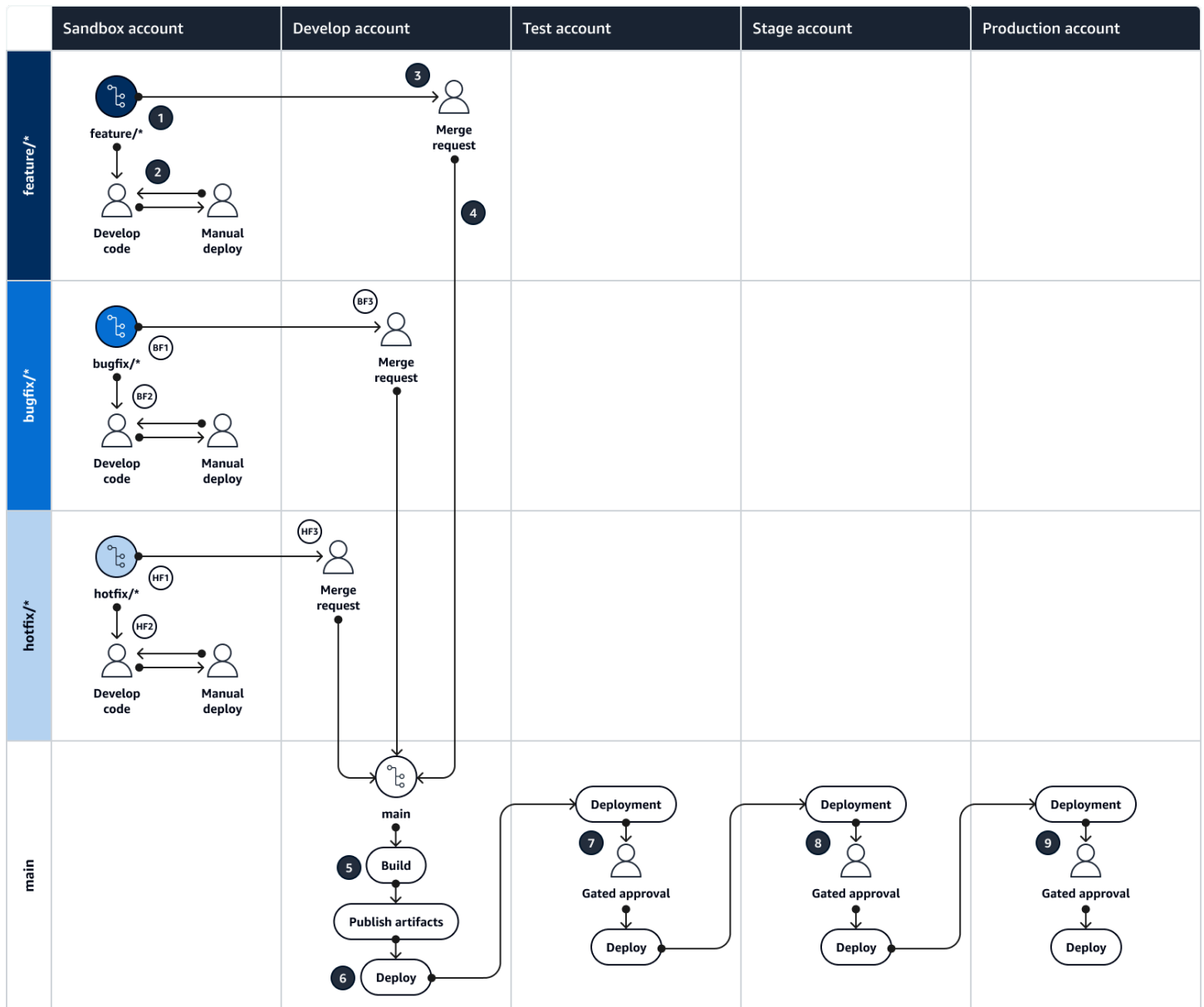
- [マルチアカウント DevOps 環境の GitHub フロー分岐戦略を実装する](#) (AWS 規範ガイド)
- [GitHub Flow Quickstart](#) (GitHub ドキュメント)
- [Why GitHub Flow?](#) (GitHub Flow ウェブサイト)

このセクションのトピック:

- [GitHub フロー戦略の視覚的な概要](#)
- [GitHub フロー戦略のブランチ](#)
- [GitHub フロー戦略の利点と欠点](#)

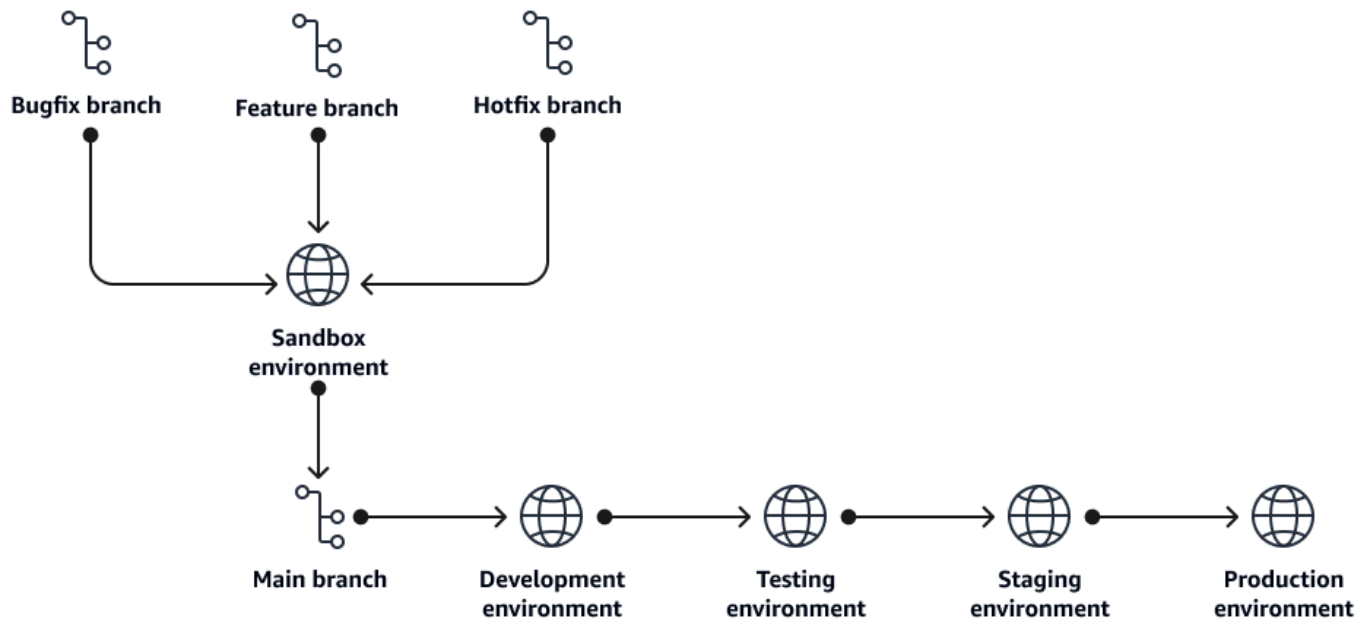
GitHub フロー戦略の視覚的な概要

次の図は、GitHub フロー分岐戦略を理解するために [Punnett の四角形](#) のように使用できます。垂直軸のブランチを水平軸の AWS 環境と並べて、各シナリオで実行するアクションを決定します。円で囲まれた数字は、図に示されている一連のアクションを案内します。この図は、サンドボックス環境の機能ブランチからメインブランチの本番リリースまで、GitHub Flow ブランチ戦略の開発ワークフローを示しています。各環境で発生するアクティビティの詳細については、このガイドの [DevOps 環境](#) を参照してください。



GitHub フロー戦略のブランチ

GitHub フロー分岐戦略には通常、次のブランチがあります。



機能ブランチ

feature ブランチで機能を開発します。feature ブランチを作成するには、main ブランチから分岐します。開発者は、feature ブランチでコードを反復、コミット、テストします。機能が完了すると、開発者はマージリクエストを作成して機能を main に昇格させます。

命名規則: `feature/<story number>_<developer initials>_<descriptor>`

命名規則の例: `feature/123456_MS_Implement _Feature_A`

バグ修正ブランチ

bugfix ブランチは問題の修正に使用されます。これらのブランチは main から分岐され、修正が完了したら main にマージされます。バグ修正がサンドボックスまたは下位の環境でテストされたら、マージリクエスト main を通じて main にマージすることで、上位の環境に昇格できます。これは整理と追跡に推奨される命名規則であり、このプロセスは機能ブランチを使用して管理することもできます。

命名規則: `bugfix/<ticket number>_<developer initials>_<descriptor>`

命名規則の例: `bugfix/123456_MS_Fix_Problem_A`

修正ブランチ

hotfix ブランチは、開発スタッフと本番環境にデプロイされたコード間の遅延を最小限に抑えながら、影響の大きい重要な問題を解決するために使用されます。これらのブランチはブランチから分岐されます。main。ホットフィックスをサンドボックスまたは下位の環境でテストしたら、マージリクエスト main を通じてにマージすることで、上位の環境に昇格できます。これは整理と追跡に推奨される命名規則であり、このプロセスは機能ブランチを使用して管理することもできます。

命名規則: `hotfix/<ticket number>_<developer initials>_<descriptor>`

命名規則の例: `hotfix/123456_MS_Fix_Problem_A`

メインブランチ

main ブランチは常に、本番環境で実行されているコードを表します。コードは、マージリクエストを使用して main ブランチから feature ブランチにマージされます。削除から保護し、デベロッパーがコードを に直接プッシュしないようにするには main、ブランチの main ブランチ保護を有効にします。

命名規則: `main`

GitHub フロー戦略の利点と欠点

GitHub フロー分岐戦略は、強力なコミュニケーションスキルを持つ、成熟した小規模な開発チームに適しています。この戦略は、継続的デリバリーを実装したいチームに最適であり、一般的な CI/CD エンジンで十分にサポートされています。GitHub Flow は軽量で、ルールが多すぎず、急速に変化するチームをサポートできます。チームが従うべき厳格なコンプライアンスまたはリリースプロセスを持っている場合、これは適していません。このモデルではマージ競合が一般的であり、頻繁に発生する可能性があります。マージの競合の解決は重要なスキルであり、それに応じてすべてのチームメンバーをトレーニングする必要があります。

利点

GitHub Flow には、開発プロセスを改善し、コラボレーションを合理化し、ソフトウェアの全体的な品質を向上させることができるいくつかの利点があります。以下は、主な利点の一部です。

- **柔軟で軽量** – GitHub Flow は、開発者がソフトウェア開発プロジェクトでコラボレーションするのに役立つ軽量で柔軟なワークフローです。これにより、最小限の複雑さで迅速な反復と実験が可能になります。
- **コラボレーションの簡素化** – GitHub Flow は、機能開発を管理するための明確で合理化されたプロセスを提供します。これにより、すばやくレビューしてマージできる小規模で焦点を絞った変更が奨励され、効率が向上します。
- **バージョン管理のクリア** – GitHub Flow では、すべての変更は別のブランチで行われます。これにより、明確で追跡可能なバージョン管理履歴が確立されます。これにより、開発者は変更を追跡して理解し、必要に応じて元に戻して、信頼性の高いコードベースを維持できます。
- **シームレスな継続的統合** – GitHub Flow は継続的統合ツールと統合されます。プルリクエストを作成すると、自動テストおよびデプロイプロセスを開始できます。CI ツールを使用すると、main ブランチにマージされる前に変更を徹底的にテストできるため、コードベースにバグを導入するリスクが軽減されます。
- **迅速なフィードバックと継続的な改善** – GitHub Flow は、プルリクエストを通じて頻繁なコードレビューと議論を促進することで、迅速なフィードバックループを促進します。これにより、問題の早期検出が容易になり、チームメンバー間の知識共有が促進され、最終的にはコード品質が向上し、開発チーム内でのコラボレーションが向上します。
- **ロールバックと復元の簡素化** – コード変更によって予期しないバグや問題が発生した場合、GitHub Flow は変更のロールバックまたは復元のプロセスを簡素化します。コミットとブランチの明確な履歴を持つことで、問題のある変更を特定して元に戻すことが容易になり、安定した機能的なコードベースを維持できます。
- **軽量学習曲線** – GitHub Flow は、特に Git とバージョン管理の概念にすでに精通しているチームにとって、Gitflow よりも簡単に学習して導入できます。そのシンプルさと直感的な分岐モデルにより、さまざまな経験レベルの開発者が利用できるようになり、新しい開発ワークフローの採用に伴う学習曲線が短縮されます。
- **継続的な開発** – GitHub Flow は、main ブランチにマージされるとすぐにすべての変更をすぐにデプロイできるようにすることで、チームが継続的なデプロイアプローチを採用できるようにします。この合理化されたプロセスにより、不要な遅延がなくなり、最新の更新と改善がユーザーに迅速に利用可能になります。これにより、開発サイクルの俊敏性と応答性が向上します。

欠点

GitHub Flow にはいくつかの利点がありますが、潜在的な欠点も考慮することが重要です。

- 大規模なプロジェクトへの適合性が限られている – GitHub Flow は、複雑なコードベースと複数の長期特徴量ブランチを持つ大規模なプロジェクトには適していない可能性があります。このような場合、Gitflow などのより構造化されたワークフローにより、同時開発とリリース管理をより適切に制御できる可能性があります。
- 正式なリリース構造の欠如 – GitHub Flow は、リリースプロセスや、バージョニング、ホットフィックス、メンテナンスブランチなどのサポート機能を明示的に定義していません。これは、厳密なリリース管理を必要とするプロジェクトや、長期的なサポートとメンテナンスを必要とするプロジェクトの制限になる可能性があります。
- 長期リリース計画のサポートが制限されている – GitHub Flow は、存続期間の短い機能ブランチに焦点を当てています。これは、厳格なロードマップや広範な機能依存関係を持つプロジェクトなど、長期リリース計画を必要とするプロジェクトとうまく一致しない可能性があります。複雑なリリーススケジュールの管理は、GitHub Flow の制約内では難しい場合があります。
- 頻繁なマージ競合の可能性 – GitHub Flow は頻繁な分岐とマージを促進するため、特に開発アクティビティの多いプロジェクトではマージ競合が発生する可能性があります。これらの競合の解決には時間がかかる場合があり、開発チームによる追加の作業が必要になる場合があります。
- 正式なワークフローフェーズの欠如 – GitHub Flow は、アルファ、ベータ、リリース候補ステージなど、開発用の明示的なフェーズを定義しません。これにより、プロジェクトの現在の状態や、さまざまなブランチやリリースの安定性レベルを伝えることが難しくなる可能性があります。
- 重大な変更の影響 – GitHub Flow は変更をmainブランチに頻繁にマージすることを奨励するため、コードベースの安定性に影響を与える重大な変更を導入するリスクが高くなります。このリスクを効果的に軽減するには、厳格なコードレビューとテストのプラクティスが不可欠です。

Gitflow 分岐戦略

Gitflow は、複数のブランチを使用して開発から本番環境にコードを移動する分岐モデルです。Gitflow は、リリースサイクルがスケジュールされており、機能のコレクションをリリースとして定義する必要があるチームに適しています。開発は、個々の機能ブランチで完了し、承認されて開発ブランチにマージされ、統合に使用されます。このブランチの機能は、本番稼働準備ができていると見なされます。計画されたすべての機能が開発ブランチに蓄積されると、上位環境にデプロイするためのリリースブランチが作成されます。この分離により、定義されたスケジュールで名前付き環境に移行する変更の制御が向上します。必要に応じて、このプロセスを高速なデプロイモデルに高速化できます。

Gitflow 分岐戦略の詳細については、次のリソースを参照してください。

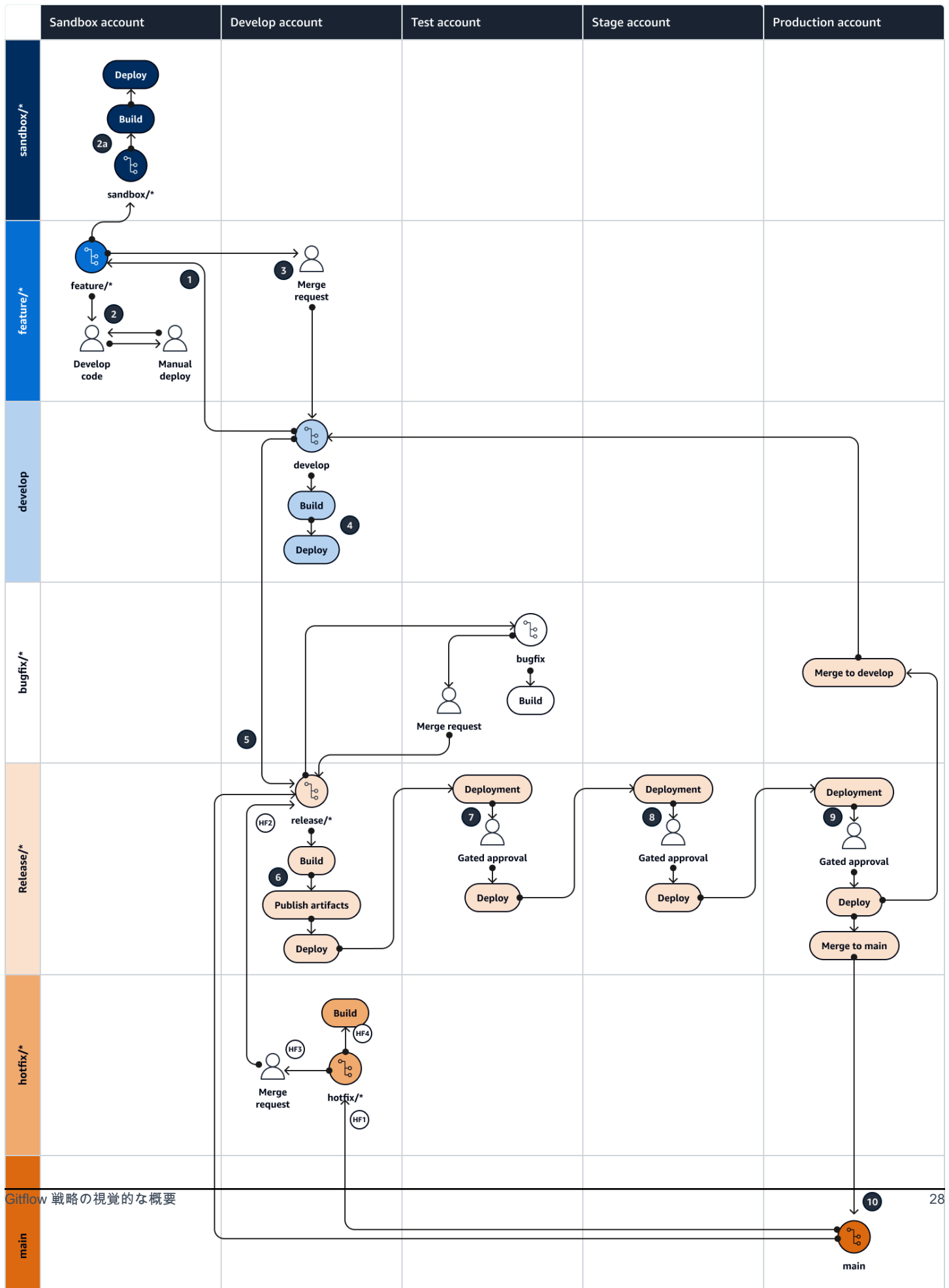
- [マルチアカウント DevOps 環境の Gitflow 分岐戦略を実装する](#) (AWS 規範ガイド)
- [元の Gitflow ブログ](#) (Vincent Driessen 氏のブログ記事)
- [Gitflow ワークフロー](#) (Atlassian)

このセクションのトピック:

- [Gitflow 戦略の視覚的な概要](#)
- [Gitflow 戦略のブランチ](#)
- [Gitflow 戦略の利点と欠点](#)

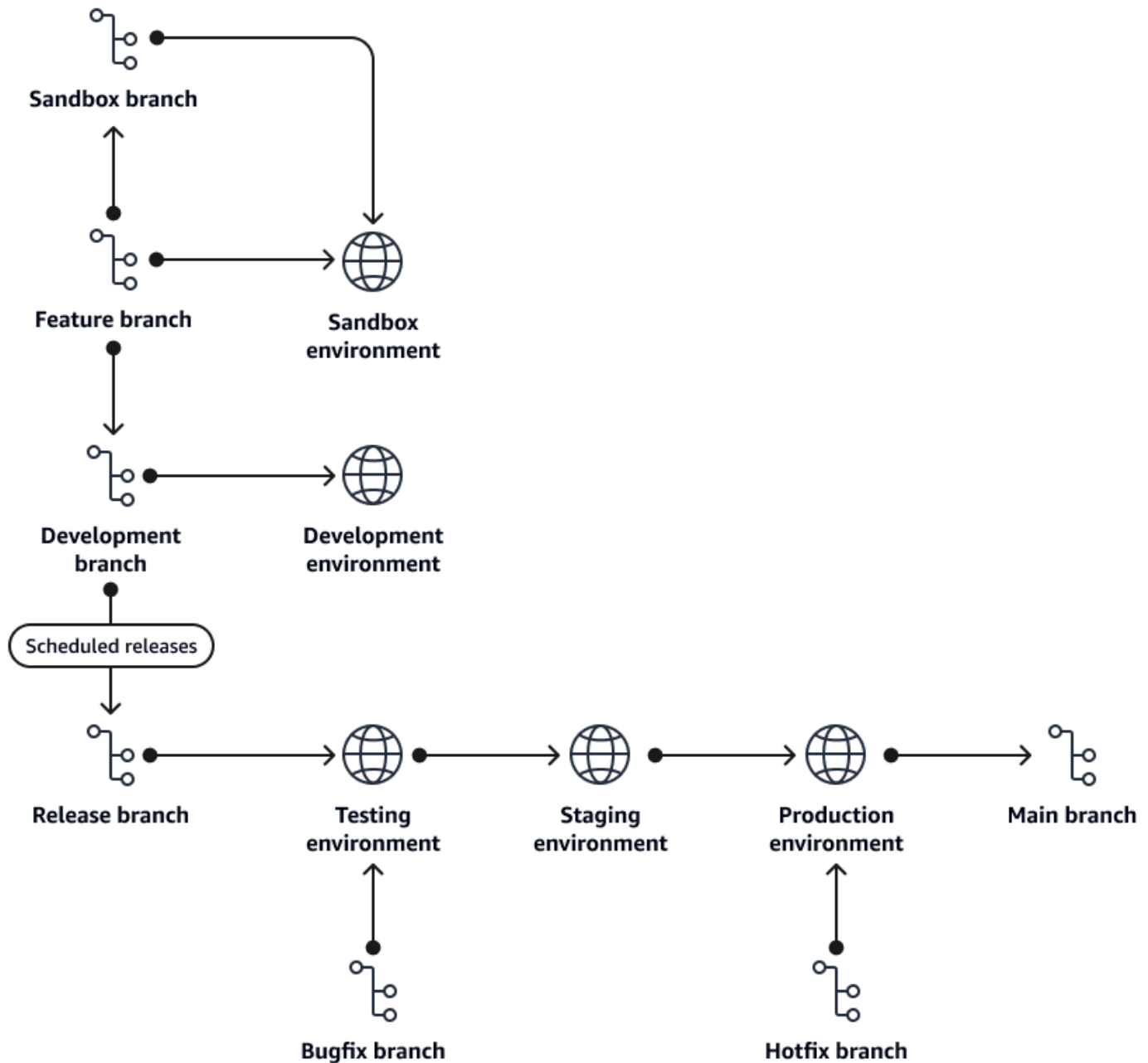
Gitflow 戦略の視覚的な概要

次の図は、Gitflow 分岐戦略を理解するために [Punnett の四角形](#) のように使用できます。垂直軸のブランチを水平軸の AWS 環境と並べて、各シナリオで実行するアクションを決定します。円で囲まれた数字は、図に示されている一連のアクションをガイドします。各環境で発生するアクティビティの詳細については、このガイドの [DevOps 環境](#) を参照してください。



Gitflow 戦略のブランチ

Gitflow 分岐戦略には通常、次のブランチがあります。



機能ブランチ

Feature ブランチは、機能を開発する短期ブランチです。feature ブランチは、develop ブランチから分岐することによって作成されます。開発者は、feature ブランチでコードを反復、コミット

ト、テストします。機能が完了すると、開発者は機能を昇格させます。特徴量ブランチから転送されるパスは 2 つだけです。

- sandbox ブランチにマージする
- develop ブランチへのマージリクエストを作成する

命名規則: feature/<story number>_<developer initials>_<descriptor>

命名規則の例: feature/123456_MS_Implement
_Feature_A

サンドボックスブランチ

sandbox ブランチは、Gitflow の非標準短期ブランチです。ただし、CI/CD パイプラインの開発に役立ちます。sandbox ブランチは主に以下の目的に使用されます。

- 手動デプロイではなく CI/CD パイプラインを使用して、サンドボックス環境への完全なデプロイを実行します。
- 開発やテストなど、より低い環境で完全なテストのためのマージリクエストを送信する前に、パイプラインを開発してテストします。

Sandbox ブランチは一時的なものであり、存続期間が長いものではありません。これらは、特定のテストが完了した後に削除する必要があります。

命名規則: sandbox/<story number>_<developer initials>_<descriptor>

命名規則の例: sandbox/123456_MS_Test_Pipe
line_Deploy

ブランチの開発

develop ブランチは、機能が統合、構築、検証され、開発環境にデプロイされる存続期間の長いブランチです。すべての feature ブランチが develop ブランチにマージされます。develop ブランチ

へのマージは、ビルドの成功と2つの開発者の承認を必要とするマージリクエストを通じて完了します。削除を防ぐには、ブランチでdevelopブランチ保護を有効にします。

命名規則: `develop`

リリースブランチ

Gitflow では、releaseブランチは短期ブランチです。これらのブランチは、ビルドオンワンのデプロイ多の方法論を取り入れて、複数の環境にデプロイできるため、特別なブランチです。Releaseブランチは、テスト環境、ステージング環境、または本番環境をターゲットにできます。開発チームがより高い環境に機能を昇格することを決定したら、新しいreleaseブランチを作成し、以前のリリースのバージョン番号をインクリメントします。各環境のゲートでは、デプロイを続けるには手動承認が必要です。Releaseブランチでは、マージリクエストを変更する必要があります。

release ブランチが本番環境にデプロイされたら、developブランチとmainブランチにマージして、バグ修正やホットフィックスが将来の開発作業にマージされるようにする必要があります。

命名規則: `release/v{major}.{minor}`

命名規則の例: `release/v1.0`

メインブランチ

main ブランチは、本番環境で実行されているコードを常に表す存続期間の長いブランチです。リリースパイプラインからのデプロイが成功すると、コードはリリースブランチからmainブランチに自動的にマージされます。削除を防ぐには、ブランチでmainブランチ保護を有効にします。

命名規則: `main`

バグ修正ブランチ

bugfix ブランチは、本番環境にリリースされていないリリースブランチの問題を修正するために使用される短期ブランチです。bugfix ブランチは、releaseブランチ内の修正をテスト、ス

ージング、または本番環境に昇格させる場合にのみ使用してください。bugfix ブランチは常に release ブランチから分岐されます。

バグ修正がテストされたら、マージリクエストを通じて release ブランチに昇格できます。その後、標準のリリースプロセスに従って release ブランチをプッシュできます。

命名規則: `bugfix/<ticket>_<developer initials>_<descriptor>`

命名規則の例: `bugfix/123456_MS_Fix_Problem_A`

修正ブランチ

hotfix ブランチは、本番環境の問題を修正するために使用される短期ブランチです。これは、本番環境に到達するために迅速化する必要がある修正を昇格させるためにのみ使用されます。hotfix ブランチは常に からブランチされます main。

ホットフィックスがテストされたら、 から作成された release ブランチへのマージリクエストを通じて本番環境に昇格できます main。テストでは、標準のリリースプロセスに従って release ブランチをプッシュできます。

命名規則: `hotfix/<ticket>_<developer initials>_<descriptor>`

命名規則の例: `hotfix/123456_MS_Fix_Problem_A`

Gitflow 戦略の利点と欠点

Gitflow 分岐戦略は、リリースとコンプライアンスの要件が厳しい大規模で分散されたチームに適しています。Gitflow は組織の予測可能なリリースサイクルに貢献します。これは多くの場合、大規模な組織で推奨されます。Gitflow は、ソフトウェア開発のライフサイクルを適切に完了するためにガードレールを必要とするチームにも適しています。これは、レビューと品質保証の機会が戦略に組み込まれているためです。Gitflow は、複数のバージョンの本番リリースを同時に維持する必要があるチームにも適しています。Gitflow の欠点は、他の分岐モデルよりも複雑であり、正常に完了するにはパターンに厳密に準拠する必要があることです。Gitflow は、リリースブランチを管理するという厳格な性質のため、継続的デリバリーを目指す組織ではうまく機能しません。Gitflow リリースブ

ランチは存続期間の長いブランチであり、タイムリーに適切に対処しないと技術的負債が蓄積される可能性があります。

利点

Gitflow ベースの開発には、開発プロセスを改善し、コラボレーションを合理化し、ソフトウェアの全体的な品質を向上させることができるいくつかの利点があります。以下は、主な利点の一部です。

- 予測可能なリリースプロセス – Gitflow は、定期的かつ予測可能なリリースプロセスに従います。これは、定期的の開発とリリースを行うチームに適しています。
- コラボレーションの向上 – Gitflow は、ブランチ feature と release ブランチの使用を推奨します。これらの 2 つのブランチは、チームが相互に最小限の依存関係で並行して作業するのに役立ちます。
- 複数の環境に最適 – Gitflow は、存続期間の長い release ブランチであるブランチを使用します。これらのブランチにより、チームは個々のリリースを長期間にわたってターゲットにすることができます。
- 本番環境の複数のバージョン – チームが本番環境のソフトウェアの複数のバージョンをサポートしている場合、Gitflow release ブランチはこの要件をサポートします。
- 組み込みコード品質レビュー – Gitflow は、コードが別の環境に昇格する前に、コードレビューと承認の使用を要求し、推奨します。このプロセスは、すべてのコードプロモーションにこのステップを要求することで、開発者間の摩擦を排除します。
- 組織のメリット – Gitflow は組織レベルでもメリットがあります。Gitflow は、組織がリリーススケジュールを理解し、予測するのに役立つ標準リリースサイクルの使用を推奨します。ビジネスは新機能をいつ配信できるかを理解しているため、配信日が設定されているため、タイムラインの摩擦が軽減されます。

欠点

Gitflow ベースの開発には、開発プロセスとチームダイナミクスに影響を与える可能性のある欠点があります。以下は、いくつかの注目すべき欠点です。

- 複雑さ – Gitflow は新しいチームが学習するための複雑なパターンであり、これを正常に使用するには Gitflow のルールに従う必要があります。
- 継続的デプロイ – Gitflow は、多くのデプロイが迅速に本番環境にリリースされるモデルには適していません。これは、Gitflow が複数のブランチを使用し、release ブランチを管理する厳格なワークフローを必要とするためです。

- ブランチ管理 – Gitflow は多くのブランチを使用するため、維持に負担がかかる場合があります。ブランチを互いに適切に整列させるために、さまざまなブランチを追跡し、リリースされたコードをマージするのは難しい場合があります。
- 技術的負債 – Gitflow リリースは通常、他の分岐モデルよりも遅いため、リリースのためにより多くの機能が蓄積され、技術的負債が蓄積される可能性があります。

チームは、Gitflow ベースの開発がプロジェクトに適したアプローチかどうかを判断する際に、これらの欠点を慎重に検討する必要があります。

次のステップ

このガイドでは、フロー、Gitflow、トランクの GitHub 3 つの一般的な Git 分岐戦略の違いについて説明します。ワークフローを詳細に説明し、それぞれの利点と欠点も示します。次のステップでは、組織の標準ワークフローのいずれかを選択します。これらの分岐戦略のいずれかを実装するには、以下を参照してください。

- [マルチアカウント DevOps 環境にトランク分岐戦略を実装する](#)
- [マルチアカウント DevOps 環境に GitHub フロー分岐戦略を実装する](#)
- [マルチアカウント DevOps 環境に Gitflow 分岐戦略を実装する](#)

Git と DevOps プロセスの使用に関するチームのジャーニーをどこから始めるべきかわからない場合は、標準ソリューションを選択してテストすることをお勧めします。標準の分岐規則を使用すると、チームは既存のドキュメントに基づいて構築し、何が最適かを学習できます。

組織や開発チームにとってうまく機能していない場合は、戦略を変更することをためらわないでください。開発チームのニーズと要件は時間の経過とともに変化する可能性があり、単一の完全なソリューションはありません。

リソース

このガイドには Git のトレーニングは含まれていませんが、このトレーニングが必要な場合は、インターネットで利用できる高品質のリソースが多数あります。[Git ドキュメント](#) サイトから始めることをお勧めします。

以下のリソースは、での Git 分岐ジャーニーに役立ちます AWS クラウド。

AWS 規範ガイド

- [マルチアカウント DevOps 環境にトランク分岐戦略を実装する](#)
- [マルチアカウント DevOps 環境に GitHub フロー分岐戦略を実装する](#)
- [マルチアカウント DevOps 環境に Gitflow 分岐戦略を実装する](#)

その他の AWS ガイド

- [AWS DevOps ガイド](#)
- [AWS デプロイパイプラインリファレンスアーキテクチャ](#)
- [DevOps とは](#)
- [DevOps リソース](#)

その他のリソース

- [12 要素のアプリ方法論](#) (12factor.net)
- [Git シークレット](#) (GitHub)
- Gitflow
 - [元の Gitflow ブログ](#) (Vincent Driessen ブログ記事)
 - [Gitflow ワークフロー](#) (アトラシアン)
 - [の Gitflow GitHub: GitHub ベースリポジトリで Git Flow ワークフローを使用する方法](#) (ビデオ) YouTube
 - [Git Flow Init の例](#) (YouTube ビデオ)
 - [Gitflow Release Branch from Start to Finish](#) (YouTube ビデオ)

- GitHub フロー
 - [GitHub フロークイックスタート](#) (GitHub ドキュメント)
 - [GitHub フローの理由](#) (GitHub Flow ウェブサイト)
- トランク
 - [「トランクベースの開発の概要」](#) (トランクベースの開発ウェブサイト)

寄稿者

オーサリング

- Mike Stephens、シニアクラウドアプリケーションアーキテクト、AWS
- Rayjan Wilson、シニアクラウドアプリケーションアーキテクト、AWS
- Abhilash Vinod、シニアクラウドアプリケーションアーキテクト、チームリード AWS

のレビュー

- Stephen DiCato、シニアセキュリティコンサルタント、AWS
- Gaurav Samudra、クラウドアプリケーションアーキテクト、AWS
- Steven Guggenheimer、シニアクラウドアプリケーションアーキテクト、チームリード AWS

テクニカルライティング

- 、 AbouHarbシニアテクニカルライター、AWS

ドキュメント履歴

以下の表は、本ガイドの重要な変更点について説明したものです。今後の更新に関する通知を受け取る場合は、[RSS フィード](#) をサブスクライブできます。

変更	説明	日付
初版発行	—	2024 年 2 月 15 日

AWS 規範ガイドの用語集

以下は、AWS 規範ガイドが提供する戦略、ガイド、パターンで一般的に使用される用語です。エントリを提案するには、用語集の最後のフィードバックの提供リンクを使用します。

数字

7 Rs

アプリケーションをクラウドに移行するための 7 つの一般的な移行戦略。これらの戦略は、ガートナーが 2011 年に特定した 5 Rs に基づいて構築され、以下で構成されています。

- リファクタリング/アーキテクチャの再設計 — クラウドネイティブ特徴を最大限に活用して、俊敏性、パフォーマンス、スケーラビリティを向上させ、アプリケーションを移動させ、アーキテクチャを変更します。これには、通常、オペレーティングシステムとデータベースの移植が含まれます。例: オンプレミスの Oracle データベースを Amazon Aurora PostgreSQL 互換エディションに移行する。
- リプラットフォーム (リフトアンドリシェイプ) — アプリケーションをクラウドに移行し、クラウド機能を活用するための最適化レベルを導入します。例: お客様のオンプレミスの Oracle データベースを AWS クラウドの Oracle 用の Amazon Relational Database Service (Amazon RDS) に移行する。
- 再購入 (ドロップアンドショップ) — 通常、従来のライセンスから SaaS モデルに移行して、別の製品に切り替えます。例: 顧客関係管理 (CRM) システムを Salesforce.com に移行する。
- リホスト (リフトアンドシフト) — クラウド機能を活用するための変更を加えずに、アプリケーションをクラウドに移行します。例: お客様のオンプレミスの Oracle データベースを AWS クラウドの EC2 インスタンス上の Oracle に移行する。
- 再配置 (ハイパーバイザーレベルのリフトアンドシフト) — 新しいハードウェアを購入したり、アプリケーションを書き換えたり、既存の運用を変更したりすることなく、インフラストラクチャをクラウドに移行できます。オンプレミスプラットフォームから同じプラットフォームのクラウドサービスにサーバーを移行します。例: Microsoft Hyper-V アプリケーションをに移行します AWS。
- 保持 (再アクセス) — アプリケーションをお客様のソース環境で保持します。これには、主要なリファクタリングを必要とするアプリケーションや、お客様がその作業を後日まで延期したいアプリケーション、およびそれらを行き移るためのビジネス上の正当性がないため、お客様が保持するレガシーアプリケーションなどがあります。
- 廃止 — お客様のソース環境で不要になったアプリケーションを停止または削除します。

A

ABAC

「[属性ベースのアクセス制御](#)」をご覧ください。

抽象化されたサービス

「[マネージドユーザー](#)」をご覧ください。

ACID

「[原子性、一貫性、分離性、耐久性 \(ACID\)](#)」をご覧ください。

アクティブ/アクティブ移行

(双方向レプリケーションツールまたは二重書き込み操作を使用して) ソースデータベースとターゲットデータベースを同期させ、移行中に両方のデータベースが接続アプリケーションからのトランザクションを処理するデータベース移行方法。この方法では、1 回限りのカットオーバーの必要がなく、管理された小規模なバッチで移行できます。[アクティブ/パッシブ移行](#)よりも柔軟な方法ですが、さらに多くの作業が必要となります。

アクティブ/パッシブ移行

ソースデータベースとターゲットデータベースを同期させながら、データがターゲットデータベースにレプリケートされている間、接続しているアプリケーションからのトランザクションをソースデータベースのみで処理するデータベース移行方法。移行中、ターゲットデータベースはトランザクションを受け付けません。

集計関数

複数行に処理を行い、グループ全体を対象に単一の戻り値を計算する SQL 関数。集計関数の例としては、SUM や MAX などがあります。

AI

「[人工知能](#)」をご覧ください。

AIOps

「[AI オペレーション](#)」をご覧ください。

匿名化

データセット内の個人情報を完全に削除するプロセス。匿名化は個人のプライバシー保護に役立ちます。匿名化されたデータは、もはや個人データとは見なされません。

アンチパターン

繰り返し起こる問題に対して頻繁に用いられる解決策で、その解決策が逆効果であったり、効果がなかったり、代替案よりも効果が低かったりするもの。

アプリケーション制御

マルウェアからシステムを保護するために、承認されたアプリケーションのみを使用できるようにするセキュリティアプローチ。

アプリケーションポートフォリオ

アプリケーションの構築と維持にかかるコスト、およびそのビジネス価値を含む、組織が使用する各アプリケーションに関する詳細情報の集まり。この情報は、[ポートフォリオの検出と分析プロセス](#)の重要な要素であり、移行、モダナイズ、最適化するアプリケーションを特定し、優先順位を付けるのに役立ちます。

人工知能 (AI)

コンピューティングテクノロジーを使用し、学習、問題の解決、パターンの認識など、通常は人間に関連づけられる認知機能の実行に特化したコンピュータサイエンスの分野。詳細については、「[人工知能 \(AI\) とは何ですか?](#)」をご覧ください。

AI オペレーション (AIOps)

機械学習技術を使用して運用上の問題を解決し、運用上のインシデントと人の介入を減らし、サービス品質を向上させるプロセス。AWS 移行戦略での AIOps の使用方法については、[オペレーション統合ガイド](#)を参照してください。

非対称暗号化

暗号化用のパブリックキーと復号用のプライベートキーから成る 1 組のキーを使用した、暗号化のアルゴリズム。パブリックキーは復号には使用されないため共有しても問題ありませんが、プライベートキーの利用は厳しく制限する必要があります。

原子性、一貫性、分離性、耐久性 (ACID)

エラー、停電、その他の問題が発生した場合でも、データベースのデータ有効性と運用上の信頼性を保証する一連のソフトウェアプロパティ。

属性ベースのアクセス制御 (ABAC)

部署、役職、チーム名など、ユーザーの属性に基づいてアクセス許可をきめ細かく設定する方法。詳細については、AWS Identity and Access Management (IAM) ドキュメントの「[の ABAC AWS](#)」を参照してください。

信頼できるデータソース

最も信頼性のある情報源とされるデータのプライマリーバージョンを保存する場所。匿名化、編集、仮名化など、データを処理または変更する目的で、信頼できるデータソースから他の場所にデータをコピーすることができます。

アベイラビリティゾーン (AZ)

他のアベイラビリティゾーンの障害から AWS リージョン 隔離され、同じリージョン内の他のアベイラビリティゾーンへの低コストで低レイテンシーのネットワーク接続を提供する 内の別の場所。

AWS クラウド導入フレームワーク (AWS CAF)

組織がクラウドへの移行を成功させるための効率的で効果的な計画を立てるための、のガイドラインとベストプラクティスのフレームワークです。AWS CAF は、ビジネス、人材、ガバナンス、プラットフォーム、セキュリティ、運用という 6 つの重点分野にガイダンスを整理しています。ビジネス、人材、ガバナンスの観点では、ビジネススキルとプロセスに重点を置き、プラットフォーム、セキュリティ、オペレーションの視点は技術的なスキルとプロセスに焦点を当てています。例えば、人材の観点では、人事 (HR)、人材派遣機能、および人材管理を扱うステークホルダーを対象としています。この観点から、AWS CAF は、クラウド導入を成功させるための組織の準備に役立つ人材開発、トレーニング、コミュニケーションのガイダンスを提供します。詳細については、[AWS CAF ウェブサイト](#)と [AWS CAF のホワイトペーパー](#) を参照してください。

AWS ワークロード認定フレームワーク (AWS WQF)

データベース移行ワークロードを評価し、移行戦略を推奨し、作業見積もりを提供するツール。AWS WQF は AWS Schema Conversion Tool (AWS SCT) に含まれています。データベーススキーマとコードオブジェクト、アプリケーションコード、依存関係、およびパフォーマンス特性を分析し、評価レポートを提供します。

B

不正なボット

個人や組織に混乱や損害を与えることを目的とした [ボット](#)。

BCP

「[ビジネス継続性計画 \(BCP\)](#)」をご覧ください。

動作グラフ

リソースの動作とインタラクションを経時的に示した、一元的なインタラクティブビュー。Amazon Detective の動作グラフを使用すると、失敗したログオンの試行、不審な API 呼び出し、その他同様のアクションを調べることができます。詳細については、Detective ドキュメントの「[動作グラフのデータ](#)」を参照してください。

ビッグエンディアンシステム

最上位バイトを最初に格納するシステム。「[エンディアン性](#)」もご覧ください。

二項分類

バイナリ結果 (2 つの可能なクラスのうちの一つ) を予測するプロセス。例えば、お客様の機械学習モデルで「この E メールはスパムですか、それともスパムではありませんか」などの問題を予測する必要があるかもしれません。または「この製品は書籍ですか、車ですか」などの問題を予測する必要があるかもしれません。

ブルームフィルター

要素がセットのメンバーであるかどうかをテストするために使用される、確率的でメモリ効率の高いデータ構造。

ブルー/グリーンデプロイ

それぞれが独立しているが、同一の環境を 2 つ作成するデプロイ戦略。現在のアプリケーションバージョンを 1 つの環境 (ブルー) で実行し、新しいアプリケーションバージョンを別の環境 (グリーン) で実行します。この戦略は、最小限の影響で迅速にロールバックするのに役立ちます。

ボット

インターネット経由で自動タスクを実行し、人間のアクティビティややり取りをシミュレートするソフトウェアアプリケーション。インターネット上の情報のインデックスを作成するウェブクロウラーなど、一部のボットは有用または有益です。悪質なボットと呼ばれる他のボットの中には、個人や組織を混乱させたり、損害を与えたりすることを意図したものもあります。

ボットネット

[マルウェア](#)に感染しており、ボットハーダーまたはボットオペレーターと呼ばれる単一の当事者によって制御されている[ボット](#)のネットワーク。ボットネットは、ボットとその影響力を拡大する仕組みとして、非常によく知られています。

ブランチ

コードリポジトリに含まれる領域。リポジトリに最初に作成するブランチは、メインブランチといます。既存のブランチから新しいブランチを作成し、その新しいブランチで機能を開発した

り、バグを修正したりできます。機能を構築するために作成するブランチは、通常、機能ブランチと呼ばれます。機能をリリースする準備ができたなら、機能ブランチをメインブランチに統合します。詳細については、「[ブランチの概要](#)」(GitHub ドキュメント)を参照してください。

ブレークグラスアクセス

例外的な状況では、承認されたプロセスを通じて、ユーザーが AWS アカウント 通常アクセス許可を持たないにすばやくアクセスできるようにします。詳細については、AWS Well-Architected ガイドの「[ブレークグラス手順の実装](#)」インジケータを参照してください。

ブラウнフィールド戦略

環境の既存インフラストラクチャ。システムアーキテクチャにブラウнフィールド戦略を導入する場合、現在のシステムとインフラストラクチャの制約に基づいてアーキテクチャを設計します。既存のインフラストラクチャを拡張している場合は、ブラウнフィールド戦略と[グリーンフィールド](#)戦略を融合させることもできます。

バッファキャッシュ

アクセス頻度が最も高いデータが保存されるメモリ領域。

ビジネス能力

価値を生み出すためにビジネスが行うこと (営業、カスタマーサービス、マーケティングなど)。マイクロサービスのアーキテクチャと開発の決定は、ビジネス能力によって推進できます。詳細については、[AWSでのコンテナ化されたマイクロサービスの実行](#)ホワイトペーパーの「[ビジネス機能を中心に組織化](#)」セクションを参照してください。

ビジネス継続性計画 (BCP)

大規模移行など、中断を伴うイベントが運用に与える潜在的な影響に対処し、ビジネスを迅速に再開できるようにする計画。

C

CAF

「[AWS クラウド導入フレームワーク](#)」を参照してください

カナリアデプロイ

エンドユーザーへのバージョンリリースを、時間をかけて段階的に行うこと。確信が持てたら新規バージョンをデプロイして、現在のバージョン全体を置き換えます。

CCoE

「[Cloud Center of Excellence](#)」を参照してください。

CDC

「[変更データキャプチャ](#)」を参照してください。

変更データキャプチャ (CDC)

データソース (データベーステーブルなど) の変更を追跡し、その変更に関するメタデータを記録するプロセス。CDC は、ターゲットシステムでの変更を監査またはレプリケートして同期を維持するなど、さまざまな目的に使用できます。

カオスエンジニアリング

障害や破壊的なイベントを意図的に導入して、システムの耐障害性をテストすること。[AWS Fault Injection Service \(AWS FIS\)](#) を使用して、AWS ワークロードにストレスを与え、その応答を評価する実験を実行できます。

CI/CD

「[継続的インテグレーションと継続的デリバリー](#)」を参照してください。

分類

予測を生成するのに役立つ分類プロセス。分類問題の機械学習モデルは、離散値を予測します。離散値は、常に互いに区別されます。例えば、モデルがイメージ内に車があるかどうかを評価する必要がある場合があります。

クライアント側の暗号化

ターゲットが AWS のサービス 受信する前に、ローカルでデータを暗号化します。

Cloud Center of Excellence (CCoE)

クラウドのベストプラクティスの作成、リソースの移動、移行のタイムラインの確立、大規模変革を通じて組織をリードするなど、組織全体のクラウド導入の取り組みを推進する学際的なチーム。詳細については、AWS クラウド エンタープライズ戦略ブログの [CCoE 投稿](#) を参照してください。

クラウドコンピューティング

リモートデータストレージと IoT デバイス管理に通常使用されるクラウドテクノロジー。クラウドコンピューティングは、一般的に、[エッジコンピューティング](#) に接続されています。

クラウド運用モデル

IT 組織において、1 つ以上のクラウド環境を構築、成熟、最適化するために使用される運用モデル。詳細については、「[クラウド運用モデルの構築](#)」を参照してください。

導入のクラウドステージ

組織が、AWS クラウドへの移行時に通常実行する 4 つの段階。

- プロジェクト — 概念実証と学習を目的として、クラウド関連のプロジェクトをいくつか実行する
- 基礎固め — お客様のクラウドの導入を拡大するための基礎的な投資 (ランディングゾーン の作成、CCoE の定義、運用モデルの確立など)
- 移行 — 個々のアプリケーションの移行
- 再発明 — 製品とサービスの最適化、クラウドでのイノベーション

これらのステージは、AWS クラウド エンタープライズ戦略ブログのブログ記事「[クラウドファーストへのジャーニー](#)」と「[導入のステージ](#)」で Stephen Orban によって定義されました。移行戦略との関連性については、AWS「[移行準備ガイド](#)」を参照してください。

CMDB

「[構成管理データベース \(CMDB\)](#)」を参照してください。

コードリポジトリ

ソースコードやその他の資産 (ドキュメント、サンプル、スクリプトなど) が保存され、バージョン管理プロセスを通じて更新される場所。一般的なクラウドリポジトリには、GitHub や Bitbucket Cloud があります。コードの各バージョンはブランチと呼ばれます。マイクロサービスの構造では、各リポジトリは 1 つの機能専用です。1 つの CI/CD パイプラインで複数のリポジトリを使用できます。

コールドキャッシュ

空である、または、かなり空きがある、もしくは、古いデータや無関係なデータが含まれているバッファキャッシュ。データベースインスタンスはメインメモリまたはディスクから読み取る必要があり、バッファキャッシュから読み取るよりも時間がかかるため、パフォーマンスに影響します。

コールドデータ

めったにアクセスされず、通常は過去のデータです。この種類のデータをクエリする場合、通常は低速なクエリでも問題ありません。このデータを低パフォーマンスで安価なストレージ階層またはクラスに移動すると、コストを削減することができます。

コンピュータビジョン (CV)

機械学習を使用してデジタルイメージやビデオといった、ビジュアル形式の情報を分析および抽出する [AI](#) の分野。例えば、Amazon SageMaker AI では、CV 用の画像処理アルゴリズムを利用できます。

設定ドリフト

ワークロードにおいて、設定が想定した状態から変化すること。これによって、ワークロードが非準拠になる可能性があります。この状態は、徐々に生じ、意図的なものではありません。

構成管理データベース (CMDB)

データベースとその IT 環境 (ハードウェアとソフトウェアの両方のコンポーネントとその設定を含む) に関する情報を保存、管理するリポジトリ。通常、CMDB のデータは、移行のポートフォリオの検出と分析の段階で使用します。

コンフォーマンスパック

コンプライアンスチェックとセキュリティチェックをカスタマイズするためにアセンブルできる AWS Config ルールと修復アクションのコレクション。YAML テンプレートを使用して、コンフォーマンスパックを AWS アカウント および リージョンの単一のエンティティとしてデプロイすることも、組織全体にデプロイすることもできます。詳細については、AWS Config ドキュメントの「[コンフォーマンスパック](#)」を参照してください。

継続的インテグレーションと継続的デリバリー (CI/CD)

ソフトウェアリリースプロセスのソース、ビルド、テスト、ステージング、本番の各ステージを自動化するプロセス。CI/CD は一般的にパイプラインと呼ばれます。プロセスの自動化、生産性の向上、コード品質の向上、配信の加速化を可能にします。詳細については、「[継続的デリバリーの利点](#)」を参照してください。CD は継続的デプロイ (Continuous Deployment) の略語でもあります。詳細については「[継続的デリバリーと継続的なデプロイ](#)」を参照してください。

CV

[「コンピュータビジョン」](#) を参照してください。

D

保管中のデータ

ストレージ内にあるデータなど、常に自社のネットワーク内にあるデータ。

データ分類

ネットワーク内のデータを重要度と機密性に基づいて識別、分類するプロセス。データに適した保護および保持のコントロールを判断する際に役立つため、あらゆるサイバーセキュリティのリスク管理戦略において重要な要素です。データ分類は、AWS Well-Architected フレームワークのセキュリティの柱のコンポーネントです。詳細については、「[データ分類](#)」を参照してください。

データドリフト

実稼働データと ML モデルのトレーニングに使用されたデータとの間に有意な差異が生じたり、入力データが時間の経過と共に有意に変化したりすることです。データドリフトは、ML モデル予測の全体的な品質、精度、公平性を低下させる可能性があります。

転送中のデータ

ネットワーク内 (ネットワークリソース間など) を活発に移動するデータ。

データメッシュ

非一元的で分散型のデータ所有権を持つとともに、一元的な管理およびガバナンスを行えるアーキテクチャフレームワーク。

データ最小化

厳密に必要なデータのみを収集し、処理するという原則。でデータ最小化を実践 AWS クラウドすることで、プライバシーリスク、コスト、分析のカーボンフットプリントを削減できます。

データ境界

AWS 環境内の一連の予防ガードレール。信頼できる ID のみが、期待されるネットワークから信頼できるリソースにアクセスできるようにします。詳細については、「[でのデータ境界の構築 AWS](#)」を参照してください。

データの前処理

raw データをお客様の機械学習モデルで簡単に解析できる形式に変換すること。データの前処理とは、特定の列または行を削除して、欠落している、矛盾している、または重複する値に対処することを意味します。

データ出所

データの生成、送信、保存の方法など、データのライフサイクル全体を通じてデータの出所と履歴を追跡するプロセス。

データ件名

データを収集、処理している個人。

データウェアハウス

分析などのビジネスインテリジェンスをサポートするデータ管理システム。データウェアハウスには、一般的に、大量の履歴データが含まれており、多くの場合、それらはクエリや分析に使用されます。

データベース定義言語 (DDL)

データベース内のテーブルやオブジェクトの構造を作成または変更するためのステートメントまたはコマンド。

データベース操作言語 (DML)

データベース内の情報を変更 (挿入、更新、削除) するためのステートメントまたはコマンド。

DDL

「[データベース定義言語](#)」を参照してください。

ディープアンサンブル

予測のために複数の深層学習モデルを組み合わせます。ディープアンサンブルを使用して、より正確な予測を取得したり、予測の不確実性を推定したりできます。

深層学習

人工ニューラルネットワークの複数層を使用して、入力データと対象のターゲット変数の間のマッピングを識別する機械学習サブフィールド。

多層防御

一連のセキュリティメカニズムとコントロールをコンピュータネットワーク全体に層状に重ねて、ネットワークとその内部にあるデータの機密性、整合性、可用性を保護する情報セキュリティの手法。この戦略を採用するときは AWS、AWS Organizations 構造の異なるレイヤーに複数のコントロールを追加して、リソースの安全性を確保します。たとえば、多層防御アプローチでは、多要素認証、ネットワークセグメンテーション、暗号化を組み合わせることができます。

委任管理者

では AWS Organizations、互換性のあるサービスが AWS メンバーアカウントを登録して組織のアカウントを管理し、そのサービスのアクセス許可を管理できます。このアカウントを、そのサービスの委任管理者と呼びます。詳細、および互換性のあるサービスの一覧は、AWS

Organizations ドキュメントの「[AWS Organizationsで利用できるサービス](#)」を参照してください。

トラブルシューティング

アプリケーション、新機能、コードの修正をターゲットの環境で利用できるようにするプロセス。デプロイでは、コードベースに変更を施した後、アプリケーションの環境でそのコードベースを構築して実行します。

開発環境

「[環境](#)」を参照してください。

検出管理

イベントが発生したときに、検出、ログ記録、警告を行うように設計されたセキュリティコントロール。これらのコントロールは副次的な防衛手段であり、実行中の予防的コントロールをすり抜けたセキュリティイベントをユーザーに警告します。詳細については、「[AWSでのセキュリティコントロールの実装](#)」の「[検出的コントロール](#)」を参照してください。

開発バリューストリームマッピング (DVSM)

ソフトウェア開発ライフサイクルのスピードと品質に悪影響を及ぼす制約を特定し、優先順位を付けるために使用されるプロセス。DVSM は、もともとリーンマニファクチャリング・プラクティスのために設計されたバリューストリームマッピング・プロセスを拡張したものです。ソフトウェア開発プロセスを通じて価値を創造し、動かすために必要なステップとチームに焦点を当てています。

デジタルツイン

建物、工場、産業機器、生産ラインなど、現実世界のシステムを仮想的に表現したものです。デジタルツインは、予知保全、リモートモニタリング、生産最適化をサポートします。

ディメンションテーブル

[スタースキーマ](#)において、ファクトテーブルの定量データに関するデータ属性が含まれる小さいテーブル。ディメンションテーブルの属性は、通常、テキストフィールド、またはテキストのように扱える個別の数値で示されます。これらの属性は、一般的に、クエリの制約、フィルタリング、結果セットのラベル付けに使用されます。

デザスタ

ワークロードまたはシステムが、導入されている主要な場所でのビジネス目標の達成を妨げるイベント。これらのイベントは、自然災害、技術的障害、または意図しない設定ミスやマルウェア攻撃などの人間の行動の結果である場合があります。

ディザスタリカバリ (DR)

[ディザスタ](#)によるダウンタイムとデータ損失を最小限に抑えるための戦略とプロセス。詳細については、AWS Well-Architected フレームワークの「[Disaster Recovery of Workloads on AWS: Recovery in the Cloud](#)」を参照してください。

DML

「[データベース操作言語](#)」を参照してください。

ドメイン駆動型設計

各コンポーネントが提供している変化を続けるドメイン、またはコアビジネス目標にコンポーネントを接続して、複雑なソフトウェアシステムを開発するアプローチ。この概念は、エリック・エヴァンスの著書、Domain-Driven Design: Tackling Complexity in the Heart of Software (ドメイン駆動設計:ソフトウェアの中心における複雑さへの取り組み) で紹介されています (ポストン: Addison-Wesley Professional、2003)。strangler fig パターンでドメイン駆動型設計を使用する方法の詳細については、「[コンテナと Amazon API Gateway を使用して、従来の Microsoft ASP.NET \(ASMX\) ウェブサービスを段階的にモダナイズ](#)」を参照してください。

DR

「[ディザスタリカバリ](#)」を参照してください。

ドリフト検出

ベースライン設定からの偏差を追跡します。たとえば、AWS CloudFormation を使用して[システムリソースのドリフトを検出](#)したり、を使用して AWS Control Tower、ガバナンス要件への準拠に影響する[ランディングゾーンの変更を検出](#)したりできます。

DVSM

「[開発バリューSTREAMマッピング](#)」を参照してください。

E

EDA

「[探索的データ分析](#)」を参照してください。

EDI

「[電子データ交換](#)」を参照してください。

エッジコンピューティング

IoT ネットワークのエッジにあるスマートデバイスの計算能力を高めるテクノロジー。[クラウドコンピューティング](#)と比較すると、エッジコンピューティングは通信レイテンシーを短縮し、応答時間を改善できます。

電子データ交換 (EDI)

組織間で行う、ビジネスドキュメントの自動交換。詳細については、[「電子データ交換とは」](#)を参照してください。

暗号化

人間が読み取り可能なプレーンテキストデータを暗号文に変換するコンピューティング処理。

暗号化キー

暗号化アルゴリズムが生成した、ランダム化されたビットからなる暗号文字列。キーの長さは決まっておらず、各キーは予測できないように、一意になるように設計されています。

エンディアン

コンピュータメモリにバイトが格納される順序。ビッグエンディアンシステムでは、最上位バイトが最初に格納されます。リトルエンディアンシステムでは、最下位バイトが最初に格納されます。

エンドポイント

[「サービスエンドポイント」](#)を参照してください。

エンドポイントサービス

仮想プライベートクラウド (VPC) 内でホストして、他のユーザーと共有できるサービス。を使用してエンドポイントサービスを作成し AWS PrivateLink、他の AWS アカウント または AWS Identity and Access Management (IAM) プリンシパルにアクセス許可を付与できます。これらのアカウントまたはプリンシパルは、インターフェイス VPC エンドポイントを作成することで、エンドポイントサービスにプライベートに接続できます。詳細については、Amazon Virtual Private Cloud (Amazon VPC) ドキュメントの [「エンドポイントサービスを作成する」](#)を参照してください。

エンタープライズリソースプランニング (ERP)

エンタープライズの主要なビジネスプロセス (会計、[MES](#)、プロジェクト管理など) を自動化および管理するシステム。

エンベロープ暗号化

暗号化キーを、別の暗号化キーを使用して暗号化するプロセス。詳細については、AWS Key Management Service (AWS KMS) ドキュメントの「[エンベロープ暗号化](#)」を参照してください。

環境

実行中のアプリケーションのインスタンス。クラウドコンピューティングにおける一般的な環境の種類は以下のとおりです。

- 開発環境 — アプリケーションのメンテナンスを担当するコアチームのみが利用できる、実行中のアプリケーションのインスタンス。開発環境は、上位の環境に昇格させる変更をテストするときに使用します。このタイプの環境は、テスト環境と呼ばれることもあります。
- 下位環境 — 初期ビルドやテストに使用される環境など、アプリケーションのすべての開発環境。
- 本番環境 — エンドユーザーがアクセスできる、実行中のアプリケーションのインスタンス。CI/CD パイプラインでは、本番環境が最後のデプロイ環境になります。
- 上位環境 — コア開発チーム以外のユーザーがアクセスできるすべての環境。これには、本番環境、本番前環境、ユーザー承認テスト環境などが含まれます。

エピック

アジャイル方法論で、お客様の作業の整理と優先順位付けに役立つ機能カテゴリ。エピックでは、要件と実装タスクの概要についてハイレベルな説明を提供します。例えば、AWS CAF セキュリティエピックには、ID とアクセスの管理、検出コントロール、インフラストラクチャセキュリティ、データ保護、インシデント対応が含まれます。AWS 移行戦略のエピックの詳細については、[プログラム実装ガイド](#)を参照してください。

ERP

「[エンタープライズリソース計画](#)」を参照してください。

探索的データ分析 (EDA)

データセットを分析してその主な特性を理解するプロセス。お客様は、データを収集または集計してから、パターンの検出、異常の検出、および前提条件のチェックのための初期調査を実行します。EDA は、統計の概要を計算し、データの可視化を作成することによって実行されます。

F

ファクトテーブル

[スタースキーマ](#)の中央にあるテーブル。ビジネスオペレーションに関する定量的データが保存されます。一般的に、ファクトテーブルは、2種類の列で構成されます。1つは測定値が含まれる列、もう1つはディメンションテーブルへの外部キーが含まれる列です。

フェイルファスト

開発ライフサイクルを短縮するために、頻繁かつ段階的にテストを行う哲学であり、アジャイルアプローチでは、この考え方がきわめて重要です。

障害分離境界

では AWS クラウド、アベイラビリティーゾーン AWS リージョン、コントロールプレーン、データプレーンなどの境界で、障害の影響を制限し、ワークロードの耐障害性を向上させるのに役立ちます。詳細については、「[AWS 障害分離境界](#)」を参照してください。

機能ブランチ

「[ブランチ](#)」を参照してください。

特徴量

お客様が予測に使用する入力データ。例えば、製造コンテキストでは、特徴量は製造ラインから定期的にキャプチャされるイメージの可能性もあります。

特徴量重要度

モデルの予測に対する特徴量の重要性。これは通常、Shapley Additive Deskonations (SHAP) や積分勾配など、さまざまな手法で計算できる数値スコアで表されます。詳細については、「[を使用した機械学習モデルの解釈可能性 AWS](#)」を参照してください。

機能変換

追加のソースによるデータのエンリッチ化、値のスケーリング、単一のデータフィールドからの複数の情報セットの抽出など、機械学習プロセスのデータを最適化すること。これにより、機械学習モデルはデータの恩恵を受けることができます。例えば、「2021-05-27 00:15:37」の日付を「2021年」、「5月」、「木」、「15」に分解すると、学習アルゴリズムがさまざまなデータコンポーネントに関連する微妙に異なるパターンを学習するのに役立ちます。

数ショットプロンプト

[LLM](#) に、タスクと望ましい出力を示す例を少数提示した後に、類似のタスクを実行させること。この手法は、プロンプトに記述された例(ショット)からモデルが学習する「インコンテキスト学

習」の一種です。数ショットプロンプトは、特定のフォーマット、推論、専門知識が必要なタスクに効果的です。「[ゼロショットプロンプト](#)」も参照してください。

FGAC

「[きめ細かなアクセス制御](#)」を参照してください。

きめ細かなアクセス制御 (FGAC)

複数の条件を使用してアクセス要求を許可または拒否すること。

フラッシュカット移行

[変更データのキャプチャ](#)による継続的なデータ複製を利用して、段階的なアプローチではなく、可能な限り短時間でデータを移行するデータベース移行方法。目的はダウンタイムを最小限に抑えることです。

FM

「[基盤モデル](#)」を参照してください。

基盤モデル (FM)

大規模な深層学習ニューラルネットワークであり、一般化およびラベル付けされていないデータからなる大規模データセットでトレーニングされています。FMにより、言語理解、テキストおよび画像生成、自然言語での会話といった、一般的な各種タスクを実行できます。詳細については、「[基盤モデルとは何ですか?](#)」を参照してください。

G

生成 AI

[AI](#) モデルのサブセット。大量のデータでトレーニングされており、シンプルなテキストプロンプトを使用して、画像、動画、テキスト、オーディオなどの新しいコンテンツやアーティファクトを作成できます。詳細については、「[生成 AI とは何ですか?](#)」を参照してください。

ジオブロッキング

「[地理的制限](#)」を参照してください。

地理的制限 (ジオブロッキング)

特定の国のユーザーがコンテンツ配信にアクセスできないようにするための、Amazon CloudFront のオプション。アクセスを許可する国と禁止する国は、許可リストまたは禁止リスト

を使って指定します。詳細については、CloudFront ドキュメントの「[コンテンツの地理的ディストリビューションの制限](#)」を参照してください。

Gitflow ワークフロー

下位環境と上位環境が、ソースコードリポジトリでそれぞれ異なるブランチを使用する方法。Gitflow ワークフローは古いと見なされている方法であり、[トランクベースのワークフロー](#)は推奨されている新しい方法です。

ゴールデンイメージ

システムまたはソフトウェアのスナップショットであり、システムまたはソフトウェアの新規インスタンスをデプロイするテンプレートとして使用されます。製造の例で言えば、ゴールデンイメージを使用すると、複数のデバイスにソフトウェアをプロビジョニングして、デバイス製造オペレーションの速度、スケーラビリティ、生産性を向上させることができます。

グリーンフィールド戦略

新しい環境に既存のインフラストラクチャが存在しないこと。システムアーキテクチャにグリーンフィールド戦略を導入する場合、既存のインフラストラクチャ (別名 [ブラウンフィールド](#)) との互換性の制約を受けることなく、あらゆる新しいテクノロジーを選択できます。既存のインフラストラクチャを拡張している場合は、ブラウンフィールド戦略とグリーンフィールド戦略を融合させることもできます。

ガードレール

組織単位 (OU) 全般のリソース、ポリシー、コンプライアンスを管理するのに役立つ概略的なルール。予防ガードレールは、コンプライアンス基準に一致するようにポリシーを実施します。これらは、サービスコントロールポリシーと IAM アクセス許可の境界を使用して実装されます。検出ガードレールは、ポリシー違反やコンプライアンス上の問題を検出し、修復のためのアラートを発信します。これらは AWS Config、AWS Security Hub CSPM、Amazon GuardDuty、AWS Trusted Advisor Amazon Inspector、およびカスタム AWS Lambda チェックを使用して実装されます。

H

HA

「[高可用性](#)」を参照してください。

異種混在データベースの移行

別のデータベースエンジンを使用するターゲットデータベースへお客様の出典データベースの移行 (例えば、Oracle から Amazon Aurora)。異種間移行は通常、アーキテクチャの再設計作業の一部であり、スキーマの変換は複雑なタスクになる可能性があります。[AWS は、スキーマの変換に役立つ AWS SCTを提供します。](#)

高可用性 (HA)

課題や災害が発生した場合に、介入なしにワークロードを継続的に運用できること。HA システムは、自動的にフェイルオーバーし、一貫して高品質のパフォーマンスを提供し、パフォーマンスへの影響を最小限に抑えながらさまざまな負荷や障害を処理するように設計されています。

ヒストリアンのモダナイゼーション

製造業のニーズによりよく応えるために、オペレーションテクノロジー (OT) システムをモダナイズし、アップグレードするためのアプローチ。ヒストリアンは、工場内のさまざまなソースからデータを収集して保存するために使用されるデータベースの一種です。

ホールドアウトデータ

[機械学習](#)モデルのトレーニング用データセットから保留される、ラベル付き履歴データの一部。ホールドアウトデータを使用すると、モデル予測をホールドアウトデータと比較して、モデルのパフォーマンスを評価できます。

同種データベースの移行

お客様の出典データベースを、同じデータベースエンジンを共有するターゲットデータベース (Microsoft SQL Server から Amazon RDS for SQL Server など) に移行する。同種間移行は、通常、リホストまたはリプラットフォーム化の作業の一部です。ネイティブデータベースユーティリティを使用して、スキーマを移行できます。

ホットデータ

リアルタイムデータや最近の翻訳データなど、頻繁にアクセスされるデータ。通常、このデータには高速なクエリ応答を提供する高性能なストレージ階層またはクラスが必要です。

ホットフィックス

本番環境の重大な問題を修正するために緊急で配布されるプログラム。緊急性が高いため、通常の DevOps のリリースワークフローからは外れた形で実施されます。

ハイパーケア期間

カットオーバー直後、移行したアプリケーションを移行チームがクラウドで管理、監視して問題に対処する期間。通常、この期間は 1~4 日です。ハイパーケア期間が終了すると、アプリケーションに対する責任は一般的に移行チームからクラウドオペレーションチームに移ります。

I

laC

「[Infrastructure as Code](#)」を参照してください。

ID ベースのポリシー

AWS クラウド 環境内のアクセス許可を定義する 1 つ以上の IAM プリンシパルにアタッチされたポリシー。

アイドル状態のアプリケーション

90 日間の平均的な CPU およびメモリ使用率が 5~20% のアプリケーション。移行プロジェクトでは、これらのアプリケーションを廃止するか、オンプレミスに保持するのが一般的です。

IIoT

「[インダストリアル IoT](#)」を参照してください。

イミュータブルインフラストラクチャ

既存インフラストラクチャの更新、パッチ適用、変更などを行わずに、本番環境ワークロードに使用する新規インフラストラクチャをデプロイするモデル。本質的に、イミュータブルインフラストラクチャは、[ミュータブルインフラストラクチャ](#)よりも一貫性、信頼性、予測性に優れています。詳細については、AWS Well-Architected フレームワークにある「[イミュータブルインフラストラクチャを使用してデプロイする](#)」のベストプラクティスを参照してください。

インバウンド (受信) VPC

AWS マルチアカウントアーキテクチャでは、アプリケーションの外部からネットワーク接続を受け入れ、検査し、ルーティングする VPC。[AWS Security Reference Architecture](#) では、アプリケーションとより広範なインターネット間の双方向のインターフェイスを保護するために、インバウンド、アウトバウンド、インスペクションの各 VPC を使用してネットワークアカウントを設定することを推奨しています。

増分移行

アプリケーションを 1 回ですべてカットオーバーするのではなく、小さい要素に分けて移行するカットオーバー戦略。例えば、最初は少数のマイクロサービスまたはユーザーのみを新しいシステムに移行する場合があります。すべてが正常に機能することを確認できたら、残りのマイクロサービスやユーザーを段階的に移行し、レガシーシステムを廃止できるようにします。この戦略により、大規模な移行に伴うリスクが軽減されます。

インダストリー 4.0

2016 年に [Klaus Schwab](#) 氏が提唱した用語で、接続、リアルタイムデータ、オートメーション、分析、AI/ML の進歩による、ビジネスプロセスのモダナイズを意味します。

インフラストラクチャ

アプリケーションの環境に含まれるすべてのリソースとアセット。

Infrastructure as Code (IaC)

アプリケーションのインフラストラクチャを一連の設定ファイルを使用してプロビジョニングし、管理するプロセス。IaC は、新しい環境を再現可能で信頼性が高く、一貫性のあるものにするため、インフラストラクチャを一元的に管理し、リソースを標準化し、スケールを迅速に行えるように設計されています。

インダストリアル IoT (IIoT)

製造、エネルギー、自動車、ヘルスケア、ライフサイエンス、農業などの産業部門におけるインターネットに接続されたセンサーやデバイスの使用。詳細については、「[インダストリアル IoT \(IIoT\) デジタルトランスフォーメーション戦略の構築](#)」を参照してください。

インスペクション VPC

AWS マルチアカウントアーキテクチャでは、VPC (同一または異なる 内 AWS リージョン)、インターネット、オンプレミスネットワーク間のネットワークトラフィックの検査を管理する一元化された VPCs。 [AWS Security Reference Architecture](#) では、アプリケーションとより広範なインターネット間の双方向のインターフェイスを保護するために、インバウンド、アウトバウンド、インスペクションの各 VPC を使用してネットワークアカウントを設定することを推奨しています。

IoT

インターネットまたはローカル通信ネットワークを介して他のデバイスやシステムと通信する、センサーまたはプロセッサが組み込まれた接続済み物理オブジェクトのネットワーク。詳細については、「[IoT とは](#)」を参照してください。

解釈可能性

機械学習モデルの特性で、モデルの予測がその入力にどのように依存するかを人間が理解できる度合いを表します。詳細については、[「を使用した機械学習モデルの解釈可能性 AWS」](#)を参照してください。

IoT

[「IoT」](#)を参照してください。

IT 情報ライブラリ (ITIL)

IT サービスを提供し、これらのサービスをビジネス要件に合わせるための一連のベストプラクティス。ITIL は ITSM の基盤を提供します。

IT サービス管理 (ITSM)

組織の IT サービスの設計、実装、管理、およびサポートに関連する活動。クラウドオペレーションと ITSM ツールの統合については、[オペレーション統合ガイド](#)を参照してください。

ITIL

[「IT 情報ライブラリ」](#)を参照してください。

ITSM

[「IT サービス管理」](#)を参照してください。

L

ラベルベースアクセス制御 (LBAC)

強制アクセス制御 (MAC) の実装で、ユーザーとデータ自体にそれぞれセキュリティラベル値が明示的に割り当てられます。ユーザーセキュリティラベルとデータセキュリティラベルが交差する部分によって、ユーザーに表示される行と列が決まります。

ランディングゾーン

ランディングゾーンは、スケーラブルで安全な、適切に設計されたマルチアカウント AWS 環境です。これは、組織がセキュリティおよびインフラストラクチャ環境に自信を持ってワークロードとアプリケーションを迅速に起動してデプロイできる出発点です。ランディングゾーンの詳細については、[「安全でスケーラブルなマルチアカウント AWS 環境のセットアップ」](#)を参照してください。

大規模言語モデル (LLM)

大量のデータで事前トレーニングされた深層学習 AI モデル。LLM では、質問への回答、ドキュメントの要約、他言語へのテキスト翻訳、文を完成させるなど、さまざまなタスクを実行できます。詳細については、「[大規模言語モデル \(LLM\) とは何ですか?](#)」を参照してください。

大規模な移行

300 台以上のサーバの移行。

LBAC

「[ラベルベースアクセス制御](#)」を参照してください。

最小特権

タスクの実行には必要最低限の権限を付与するという、セキュリティのベストプラクティス。詳細については、IAM ドキュメントの「[最小特権アクセス許可を適用する](#)」を参照してください。

リフトアンドシフト

「[7 Rs](#)」を参照してください。

リトルエンディアンシステム

最下位バイトを最初に格納するシステム。「[エンディアン性](#)」もご覧ください。

LLM

「[大規模言語モデル](#)」を参照してください。

下位環境

「[環境](#)」を参照してください。

M

機械学習 (ML)

パターン認識と学習にアルゴリズムと手法を使用する人工知能の一種。ML は、モノのインターネット (IoT) データなどの記録されたデータを分析して学習し、パターンに基づく統計モデルを生成します。詳細については、「[機械学習](#)」を参照してください。

メインブランチ

「[ブランチ](#)」を参照してください。

マルウェア

コンピュータのセキュリティやプライバシーを侵害するように設計されたソフトウェア。マルウェアは、コンピュータシステムの中断、機密情報の漏洩、不正アクセスを招く可能性があります。マルウェアの例には、ウイルス、ワーム、ランサムウェア、トロイの木馬、スパイウェア、キーロガーなどがあります。

マネージドサービス

AWS のサービスはインフラストラクチャレイヤー、オペレーティングシステム、プラットフォーム AWS を運用し、エンドポイントにアクセスしてデータを保存および取得します。マネージドサービスの例として、Amazon Simple Storage Service (Amazon S3) と Amazon DynamoDB が挙げられます。このサービスは、抽象化されたサービスとも呼ばれます。

製造実行システム (MES)

生産プロセスを追跡、モニタリング、文書化、制御するソフトウェアシステムであり、工場では、これによって、原材料から製品を完成させます。

MAP

[「Migration Acceleration Program」](#) を参照してください。

メカニズム

ツールを作成してその導入を推進し、導入結果を調べて調整を行うための包括的なプロセス。メカニズムとは、運用中にそれ自体を強化し改善するサイクルを意味します。詳細については、AWS 「Well-Architected フレームワーク」の [「メカニズムの構築」](#) を参照してください。

メンバーアカウント

組織の一部である管理アカウント AWS アカウント 以外のすべて AWS Organizations。アカウントが組織のメンバーになることができるのは、一度に 1 つのみです。

MES

[「製造実行システム」](#) を参照してください。

Message Queuing Telemetry Transport (MQTT)

[発行/サブスクリプション](#) のパターンに基づく、軽量のマシンツーマシン (M2M) 通信プロトコルであり、リソースに限りのある [IoT](#) デバイスに使用されます。

マイクロサービス

明確に定義された API を介して通信し、通常は小規模な自己完結型のチームが所有する、小規模で独立したサービスです。例えば、保険システムには、販売やマーケティングなどのビジネス

機能、または購買、請求、分析などのサブドメインにマッピングするマイクロサービスが含まれる場合があります。マイクロサービスの利点には、俊敏性、柔軟なスケーリング、容易なデプロイ、再利用可能なコード、回復力などがあります。詳細については、[AWS「サーバーレスサービスを使用したマイクロサービスの統合」](#)を参照してください。

マイクロサービスアーキテクチャ

各アプリケーションプロセスをマイクロサービスとして実行する独立したコンポーネントを使用してアプリケーションを構築するアプローチ。これらのマイクロサービスは、軽量 API を使用して、明確に定義されたインターフェイスを介して通信します。このアーキテクチャの各マイクロサービスは、アプリケーションの特定の機能に対する需要を満たすように更新、デプロイ、およびスケーリングできます。詳細については、「[でのマイクロサービスの実装 AWS](#)」を参照してください。

Migration Acceleration Program (MAP)

組織がクラウドに移行するための強力な運用基盤を構築し、移行の初期コストを相殺するのに役立つコンサルティングサポート、トレーニング、サービスを提供する AWS プログラム。MAP には、組織的な方法でレガシー移行を実行するための移行方法論と、一般的な移行シナリオを自動化および高速化する一連のツールが含まれています。

大規模な移行

アプリケーションポートフォリオの大部分を次々にクラウドに移行し、各ウェーブでより多くのアプリケーションを高速に移動させるプロセス。この段階では、以前の段階から学んだベストプラクティスと教訓を使用して、移行ファクトリー チーム、ツール、プロセスのうち、オートメーションとアジャイルデリバリーによってワークロードの移行を合理化します。これは、[AWS 移行戦略](#) の第 3 段階です。

移行ファクトリー

自動化された俊敏性のあるアプローチにより、ワークロードの移行を合理化する部門横断的なチーム。移行ファクトリーチームには、通常、運用、ビジネスアナリストおよび所有者、移行エンジニア、デベロッパー、およびスプリントで作業する DevOps プロフェッショナルが含まれます。エンタープライズアプリケーションポートフォリオの 20~50% は、ファクトリーのアプローチによって最適化できる反復パターンで構成されています。詳細については、このコンテンツセットの[移行ファクトリーに関する解説](#)と [Cloud Migration Factory ガイド](#)を参照してください。

移行メタデータ

移行を完了するために必要なアプリケーションおよびサーバーに関する情報。移行パターンごとに、異なる一連の移行メタデータが必要です。移行メタデータの例としては、ターゲットサブネット、セキュリティグループ、AWS アカウントなどがあります。

移行パターン

移行戦略、移行先、および使用する移行アプリケーションまたはサービスを詳述する、反復可能な移行タスク。例: AWS Application Migration Service を使用して Amazon EC2 への移行をリホストします。

Migration Portfolio Assessment (MPA)

オンラインツール。これによって、AWS クラウドに移行するビジネスケースの検証に必要な情報を得られます。MPA は、詳細なポートフォリオ評価 (サーバーの適切なサイジング、価格設定、TCO 比較、移行コスト分析) および移行プラン (アプリケーションデータの分析とデータ収集、アプリケーションのグループ化、移行の優先順位付け、およびウェーブプランニング) を提供します。[MPA ツール](#) (ログインが必要) は、すべての AWS コンサルタントと APN パートナー コンサルタントが無料で利用できます。

移行準備状況評価 (MRA)

AWS CAF を使用して、組織のクラウド準備状況に関するインサイトを取得し、長所と短所を特定し、特定されたギャップを埋めるためのアクションプランを構築するプロセス。詳細については、[移行準備状況ガイド](#)を参照してください。MRA は、[AWS 移行戦略](#)の第一段階です。

移行戦略

ワークロードを AWS クラウドに移行するために使用するアプローチ。詳細については、この用語集の [7 Rs](#) エントリと、「[組織を動員して大規模な移行を加速する](#)」を参照してください。

ML

「[機械学習](#)」を参照してください。

モダナイゼーション

古い (レガシーまたはモノリシック) アプリケーションとそのインフラストラクチャをクラウド内の俊敏で弾力性のある高可用性システムに変換して、コストを削減し、効率を高め、イノベーションを活用します。詳細については、「[AWS クラウドでのアプリケーションのモダナイズ戦略](#)」を参照してください。

モダナイゼーション準備状況評価

組織のアプリケーションのモダナイゼーションの準備状況を判断し、利点、リスク、依存関係を特定し、組織がこれらのアプリケーションの将来の状態をどの程度適切にサポートできるかを決定するのに役立つ評価。評価の結果として、ターゲットアーキテクチャのブループリント、モダナイゼーションプロセスの開発段階とマイルストーンを詳述したロードマップ、特定されたギャップに対処するためのアクションプランが得られます。詳細については、「[AWS クラウドでのアプリケーションのモダナイゼーションの準備状況を評価する](#)」を参照してください。

モノリシックアプリケーション (モノリス)

緊密に結合されたプロセスを持つ単一のサービスとして実行されるアプリケーション。モノリシックアプリケーションにはいくつかの欠点があります。1つのアプリケーション機能エクスペリエンスの需要が急増する場合は、アーキテクチャ全体をスケーリングする必要があります。モノリシックアプリケーションの特徴を追加または改善することは、コードベースが大きくなると複雑になります。これらの問題に対処するには、マイクロサービスアーキテクチャを使用できます。詳細については、「[モノリスをマイクロサービスに分解する](#)」を参照してください。

MPA

「[Migration Portfolio Assessment](#)」を参照してください。

MQTT

「[Message Queuing Telemetry Transport](#)」を参照してください。

多クラス分類

複数のクラスの予測を生成するプロセス (2 つ以上の結果の 1 つを予測します)。例えば、機械学習モデルが、「この製品は書籍、自動車、電話のいずれですか?」または、「このお客様にとって最も関心のある商品のカテゴリはどれですか?」と聞くかもしれません。

ミュータブルなインフラストラクチャ

本番ワークロードに使用する既存のインフラストラクチャを更新および変更するためのモデル。Well-Architected AWS フレームワークでは、一貫性、信頼性、予測可能性を向上させるために、[イミュータブルインフラストラクチャ](#)の使用をベストプラクティスとして推奨しています。

O

OAC

「[オリジンアクセス制御](#)」を参照してください。

OAI

「[オリジンアクセスアイデンティティ](#)」を参照してください。

OCM

「[組織変更管理](#)」を参照してください。

オフライン移行

移行プロセス中にソースワークロードを停止させる移行方法。この方法はダウンタイムが長くなるため、通常は重要ではない小規模なワークロードに使用されます。

OI

「[オペレーション統合](#)」を参照してください。

Ola

「[オペレーショナルレベルアグリーメント](#)」を参照してください。

オンライン移行

ソースワークロードをオフラインにせずにターゲットシステムにコピーする移行方法。ワークロードに接続されているアプリケーションは、移行中も動作し続けることができます。この方法はダウンタイムがゼロから最小限で済むため、通常は重要な本番稼働環境のワークロードに使用されます。

OPC-UA

「[Open Process Communications - Unified Architecture](#)」を参照してください。

Open Process Communications - Unified Architecture (OPC-UA)

産業オートメーション用のマシンツーマシン (M2M) 通信プロトコル。OPC-UA により、相互運用の際に、データ暗号化、認証、認可の各スキームを標準化できます。

オペレーショナルレベルアグリーメント (OLA)

サービスレベルアグリーメント (SLA) をサポートするために、どの機能的 IT グループが互いに提供することを約束するかを明確にする契約。

運用準備状況レビュー (ORR)

質問と関連するベストプラクティスのチェックリスト。インシデントや起こり得る障害を理解、評価、防止したり、その範囲を縮小したりする際に役立ちます。詳細については、AWS Well-Architected フレームワークの「[Operational Readiness Reviews \(ORR\)](#)」を参照してください。

運用テクノロジー (OT)

産業オペレーション、機器、インフラストラクチャを制御するために物理環境と連携させるハードウェアおよびソフトウェアシステム。製造分野では、[Industry 4.0](#) への変革を進める上で、OT と情報技術 (IT) システムの統合に焦点が当てられています。

オペレーション統合 (OI)

クラウドでオペレーションをモダナイズするプロセスには、準備計画、オートメーション、統合が含まれます。詳細については、[オペレーション統合ガイド](#)を参照してください。

組織の証跡

組織 AWS アカウント 内のすべてのイベント AWS CloudTrail をログに記録することによって作成された証跡 AWS Organizations。証跡は、組織に含まれている各 AWS アカウントに作成され、各アカウントのアクティビティを追跡します。詳細については、CloudTrail ドキュメントの「[組織の証跡の作成](#)」を参照してください。

組織変更管理 (OCM)

人材、文化、リーダーシップの観点から、主要な破壊的なビジネス変革を管理するためのフレームワーク。OCM は、変化の導入を加速し、移行問題に対処し、文化や組織の変化を推進することで、組織が新しいシステムと戦略の準備と移行するのを支援します。AWS 移行戦略では、クラウド導入プロジェクトに必要な変化のスピードにより、このフレームワークは人材アクセラレーションと呼ばれます。詳細については、[OCM ガイド](#)を参照してください。

オリジンアクセス制御 (OAC)

Amazon Simple Storage Service (Amazon S3) コンテンツを保護するための、CloudFront のアクセス制限の強化オプション。OAC は AWS リージョン、すべての S3 バケット、AWS KMS (SSE-KMS) によるサーバー側の暗号化、S3 バケットへの動的 PUT および DELETE リクエストをサポートします。

オリジンアクセスアイデンティティ (OAI)

CloudFront の、Amazon S3 コンテンツを保護するためのアクセス制限オプション。OAI を使用すると、CloudFront が、Amazon S3 に認証可能なプリンシパルを作成します。認証されたプリンシパルは、S3 バケット内のコンテンツに、特定の CloudFront ディストリビューションを介してのみアクセスできます。[OAC](#) も併せて参照してください。OAC では、より詳細な、強化されたアクセス制御が可能です。

ORR

「[運用準備状況レビュー](#)」を参照してください。

OT

「[運用テクノロジー](#)」を参照してください。

アウトバウンド (送信) VPC

AWS マルチアカウントアーキテクチャでは、アプリケーション内から開始されたネットワーク接続を処理する VPC。[AWS Security Reference Architecture](#) では、アプリケーションとより広範なインターネット間の双方向のインターフェイスを保護するために、インバウンド、アウトバウンド、インスペクションの各 VPC を使用してネットワークアカウントを設定することを推奨しています。

P

アクセス許可の境界

ユーザーまたはロールが使用できるアクセス許可の上限を設定する、IAM プリンシパルにアタッチされる IAM 管理ポリシー。詳細については、IAM ドキュメントの[アクセス許可の境界](#)を参照してください。

個人を特定できる情報 (PII)

直接閲覧した場合、または他の関連データと組み合わせた場合に、個人の身元を合理的に推測するために使用できる情報。PII の例には、氏名、住所、連絡先情報などがあります。

PII

「[個人を特定できる情報](#)」を参照してください。

プレイブック

クラウドでのコアオペレーション機能の提供など、移行に関連する作業を取り込む、事前定義された一連のステップ。プレイブックは、スクリプト、自動ランブック、またはお客様のモダナイズされた環境を運用するために必要なプロセスや手順の要約などの形式をとることができます。

PLC

「[プログラマブルロジックコントローラー](#)」を参照してください。

PLM

「[製品ライフサイクル管理](#)」を参照してください。

ポリシー

次の操作を可能にするオブジェクト: アクセス許可を定義する ([ID ベースのポリシー](#)を参照)。アクセス条件を指定する ([リソースベースのポリシー](#)を参照)。AWS Organizations の組織における全アカウントにアクセス許可の上限を定義する ([サービスコントロールポリシー](#)を参照)。

多言語の永続性

データアクセスパターンやその他の要件に基づいて、マイクロサービスのデータストレージテクノロジーを個別に選択します。マイクロサービスが同じデータストレージテクノロジーを使用している場合、実装上の問題が発生したり、パフォーマンスが低下する可能性があります。マイクロサービスは、要件に最も適合したデータストアを使用すると、より簡単に実装でき、パフォーマンスとスケーラビリティが向上します。

ポートフォリオ評価

移行を計画するために、アプリケーションポートフォリオの検出、分析、優先順位付けを行うプロセス。詳細については、「[移行の準備状況の評価](#)」を参照してください。

述語

true または false を返すためのクエリ条件。一般的に、WHERE 句に記述されます。

述語プッシュダウン

データベースクエリを最適化する手法。これによって、転送前にクエリ内のデータをフィルタリングします。この手法を取ると、リレーショナルデータベースから取得し処理する必要のあるデータの量が減少するため、クエリのパフォーマンスが向上します。

予防的コントロール

イベントの発生を防ぐように設計されたセキュリティコントロール。このコントロールは、ネットワークへの不正アクセスや好ましくない変更を防ぐ最前線の防御です。詳細については、「AWSでのセキュリティコントロールの実装」の「[予防的コントロール](#)」を参照してください。

プリンシパル

アクションを実行し AWS、リソースにアクセスできるエンティティ。このエンティティは通常、IAM AWS アカウントロール、またはユーザーのルートユーザーです。詳細については、IAM ドキュメントの「[ロールに関する用語と概念](#)」にあるプリンシパルを参照してください。

プライバシーバイデザイン

開発プロセス全体を通してプライバシーが考慮されているシステムエンジニアリングのアプローチ。

プライベートホストゾーン

1 つ以上の VPC 内のドメインとそのサブドメインへの DNS クエリに対し、Amazon Route 53 がどのように応答するかに関する情報を保持するコンテナ。詳細については、Route 53 ドキュメントの「[プライベートホストゾーンの使用](#)」を参照してください。

プロアクティブコントロール

非準拠リソースのデプロイ防止を目的とした[セキュリティコントロール](#)。このコントロールにより、プロビジョニング前にリソースをスキャンします。コントロールに準拠していないリソースは、プロビジョニングされません。詳細については、AWS Control Tower ドキュメントの「[コントロールリファレンスガイド](#)」および「[セキュリティコントロールの実装](#)」の「[プロアクティブコントロール](#)」を参照してください。 AWS

製品ライフサイクル管理 (PLM)

製品の設計、開発、発売から、成長、成熟、衰退、廃棄に至る、製品のライフサイクル全体を通してデータとプロセスを管理すること。

本番環境

「[環境](#)」を参照してください。

プログラマブルロジックコントローラー (PLC)

製造分野で使用される、信頼性と適応性に優れたコンピュータであり、これによって、マシンをモニタリングするとともに、製造プロセスを自動化します。

プロンプトチェイニング

1 つの [LLM](#) プロンプトによる出力を次のプロンプトの入力に使用して、より良いレスポンスを生成します。この手法を使用すると、複雑なタスクをサブタスクに分割したり、事前レスポンスを繰り返し改良または拡張したりできます。これによって、モデルのレスポンスの精度と関連性が向上し、粒度の高いパーソナライズされた結果を得られます。

仮名化

データセット内の個人識別子をプレースホルダー値に置き換えるプロセス。仮名化は個人のプライバシー保護に役立ちます。仮名化されたデータは、依然として個人データとみなされます。

発行/サブスクライブ (pub/sub)

マイクロサービス間の非同期通信を可能にするパターン。これにより、スケーラビリティと応答性を向上させます。例えば、マイクロサービスベースの [MES](#) の場合、マイクロサービスは、他のマイクロサービスがサブスクライブ可能なチャンネルにイベントメッセージを発行できます。このシステムでは、発行サービスの変更なしに、新規マイクロサービスを追加できます。

Q

クエリプラン

手順などの一連のステップであり、SQL リレーショナルデータベースシステムのデータにアクセスするために使用されます。

クエリプランのリグレッション

データベースサービスのオプティマイザーが、データベース環境に特定の変更が加えられる前に選択されたプランよりも最適性の低いプランを選択すること。これは、統計、制限事項、環境設定、クエリパラメータのバインディングの変更、およびデータベースエンジンの更新などが原因である可能性があります。

R

RACI マトリックス

「[実行責任者、説明責任者、協業先、報告先 \(RACI\)](#)」を参照してください。

RAG

「[検索拡張生成](#)」を参照してください。

ランサムウェア

決済が完了するまでコンピュータシステムまたはデータへのアクセスをブロックするように設計された、悪意のあるソフトウェア。

RASCI マトリックス

「[実行責任者、説明責任者、協業先、報告先 \(RACI\)](#)」を参照してください。

RCAC

「[行と列のアクセス制御](#)」を参照してください。

リードレプリカ

読み取り専用で使用されるデータベースのコピー。クエリをリードレプリカにルーティングして、プライマリデータベースへの負荷を軽減できます。

リアーキテクト

「[7 Rs](#)」を参照してください。

目標復旧時点 (RPO)

最後のデータリカバリポイントからの最大許容時間です。これにより、最後の回復時点からサービスが中断されるまでの間に許容できるデータ損失の程度が決まります。

目標復旧時間 (RTO)

サービスが中断から復旧までの最大許容遅延時間。

リファクタリング

「[7 Rs](#)」を参照してください。

リージョン

地理的エリア内の AWS リソースのコレクション。各 AWS リージョンは、耐障害性、安定性、耐障害性を提供するために、他のから分離され、独立しています。詳細については、「[アカウントが使用できる AWS リージョンを指定する](#)」を参照してください。

リグレッション

数値を予測する機械学習手法。例えば、「この家はどれくらいの値段で売れるでしょうか?」という問題を解決するために、機械学習モデルは、線形回帰モデルを使用して、この家に関する既知の事実 (平方フィートなど) に基づいて家の販売価格を予測できます。

リホスト

「[7 Rs](#)」を参照してください。

リリース

デプロイプロセスで、変更を本番環境に昇格させること。

再配置

「[7 Rs](#)」を参照してください。

リプラットフォーム

「[7 Rs](#)」を参照してください。

再購入

「[7 Rs](#)」を参照してください。

回復性

中断に抵抗または中断から回復するアプリケーションの機能。AWS クラウドでの回復力を計画する際には、一般的に、[高可用性](#)と[ディザスタリカバリ](#)が考慮されます。詳細については、「[AWS クラウドの耐障害性](#)」を参照してください。

リソースベースのポリシー

Amazon S3 バケット、エンドポイント、暗号化キーなどのリソースにアタッチされたポリシー。このタイプのポリシーは、アクセスが許可されているプリンシパル、サポートされているアクション、その他の満たすべき条件を指定します。

実行責任者、説明責任者、協業先、報告先 (RACI) に基づくマトリックス

移行活動とクラウド運用に関わるすべての関係者の役割と責任を定義したマトリックス。マトリックスの名前は、マトリックスで定義されている責任の種類、すなわち責任 (R)、説明責任 (A)、協議 (C)、情報提供 (I) に由来します。サポート (S) タイプはオプションです。サポートが含まれる場合は RASCI マトリックスと呼ばれ、含まれない場合は RACI マトリックスと呼ばれます。

レスポンスコントロール

有害事象やセキュリティベースラインからの逸脱について、修復を促すように設計されたセキュリティコントロール。詳細については、「AWSでのセキュリティコントロールの実装」の「[レスポンスコントロール](#)」を参照してください。

保持

「[7 Rs](#)」を参照してください。

廃止

「[7 Rs](#)」を参照してください。

検索拡張生成 (RAG)

[生成 AI](#) の技術。これにより、[LLM](#) では、レスポンスの生成前に、トレーニングデータソースの外部にある信頼できるデータソースが参照されます。例えば、RAG モデルによって、組織のナレッジベースまたはカスタムデータのセマンティック検索を実行できる場合があります。細については、「[RAG \(検索拡張生成\) とは何ですか?](#)」を参照してください。

ローテーション

定期的に[シークレット情報](#)を更新して、攻撃者が認証情報にアクセスするのをより困難にするプロセス。

行と列のアクセス制御 (RCAC)

アクセスルールが定義された、基本的で柔軟な SQL 表現の使用。RCAC は行権限と列マスクで構成されています。

RPO

「[目標復旧時点](#)」を参照してください。

RTO

「[目標復旧時間](#)」を参照してください。

ランブック

特定のタスクを実行するために必要な手動または自動化された一連の手順。これらは通常、エラー率の高い反復操作や手順を合理化するために構築されています。

S

SAML 2.0

多くの ID プロバイダー (IdP) が使用しているオープンスタンダード。この機能を使用すると、フェデレーテッドシングルサインオン (SSO) が有効になるため、ユーザーは組織内のすべてのユーザーを IAM で作成しなくても、AWS マネジメントコンソールにログインしたり AWS、API オペレーションを呼び出すことができます。SAML 2.0 ベースのフェデレーションの詳細については、IAM ドキュメントの「[SAML 2.0 ベースのフェデレーションについて](#)」を参照してください。

SCADA

「[監視制御とデータ取得](#)」を参照してください。

SCP

「[サービスコントロールポリシー](#)」を参照してください。

シークレット

暗号化された形式で保存するパスワードやユーザー認証情報などの AWS Secrets Manager 機密情報または制限付き情報。シークレット値とそのメタデータで構成されます。シークレット値には、バイナリ、1 つの文字列、複数の文字列を指定できます。詳細については、Secrets Manager ドキュメントの「[Secrets Manager シークレットの概要](#)」を参照してください。

セキュリティバイデザイン

開発プロセス全体を通してセキュリティが考慮されているシステムエンジニアリングのアプローチ。

セキュリティコントロール

脅威アクターによるセキュリティ脆弱性の悪用を防止、検出、軽減するための、技術上または管理上のガードレール。セキュリティコントロールには、主に 4 つの種類があります。4 つとは、[予防](#)、[検出](#)、[レスポンス](#)、[プロアクティブ](#)です。

セキュリティ強化

アタックサーフェスを狭めて攻撃への耐性を高めるプロセス。このプロセスには、不要になったリソースの削除、最小特権を付与するセキュリティのベストプラクティスの実装、設定ファイル内の不要な機能の無効化、といったアクションが含まれています。

Security Information and Event Management (SIEM) システム

セキュリティ情報管理 (SIM) とセキュリティイベント管理 (SEM) のシステムを組み合わせたツールとサービス。SIEM システムは、サーバー、ネットワーク、デバイス、その他ソースからデータを収集、モニタリング、分析して、脅威やセキュリティ違反を検出し、アラートを発信します。

セキュリティレスポンスの自動化

セキュリティイベントへの自動レスポンスまたは自動修復を目的として、事前定義およびプログラムされたアクション。これらの自動化は、セキュリティのベストプラクティスを実装するのに役立つ[検出的](#)または[応答的](#)な AWS セキュリティコントロールとして機能します。自動レスポンスアクションの例には、VPC セキュリティグループの変更、Amazon EC2 インスタンスへのパッチ適用、認証情報の更新などがあります。

サーバー側の暗号化

送信先にあるデータの、それ AWS のサービスを受け取る による暗号化。

サービスコントロールポリシー (SCP)

AWS Organizationsの組織内の、すべてのアカウントのアクセス許可を一元的に管理するポリシー。SCP は、管理者がユーザーまたはロールに委任するアクションに、ガードレールを定義したり、アクションの制限を設定したりします。SCP は、許可リストまたは拒否リストとして、許可または禁止するサービスやアクションを指定する際に使用できます。詳細については、AWS Organizations ドキュメントの「[サービスコントロールポリシー](#)」を参照してください。

サービスエンドポイント

のエンドポイントの URL AWS のサービス。ターゲットサービスにプログラムで接続するには、エンドポイントを使用します。詳細については、「AWS 全般のリファレンス」の「[AWS のサービス エンドポイント](#)」を参照してください。

サービスレベルアグリーメント (SLA)

サービスのアップタイムやパフォーマンスなど、IT チームがお客様に提供すると約束したものを明示した合意書。

サービスレベルインジケータ (SLI)

エラー率、可用性、スループットといった、サービスパフォーマンス面の指標。

サービスレベル目標 (SLO)

[サービスレベルインジケータ](#)によって測定され、サービスの状態を表すターゲットメトリクス。

責任共有モデル

クラウドのセキュリティとコンプライアンス AWS について と共有する責任を説明するモデル。AWS はクラウドのセキュリティを担当しますが、お客様はクラウドのセキュリティを担当します。詳細については、「[責任共有モデル](#)」を参照してください。

SIEM

「[Security Information and Event Management システム](#)」を参照してください。

単一障害点 (SPOF)

特定のアプリケーションを構成する単一の重要なコンポーネントで発生し、システム稼働に支障をきたす可能性のある障害。

SLA

「[サービスレベルアグリーメント](#)」を参照してください。

SLI

「[サービスレベルインジケータ](#)」を参照してください。

SLO

「[サービスレベルの目標](#)」を参照してください。

スプリットアンドシードモデル

モダナイゼーションプロジェクトのスケーリングと加速のためのパターン。新機能と製品リリースが定義されると、コアチームは解放されて新しい製品チームを作成します。これにより、お客様の組織の能力とサービスの拡張、デベロッパーの生産性の向上、迅速なイノベーションのサポートに役立ちます。詳細については、「[AWS クラウドでのアプリケーションをモダナイズするための段階的アプローチ](#)」を参照してください。

SPOF

「[単一障害点](#)」を参照してください。

スタースキーマ

データベースの編成構造を意味し、1つの大きいファクトテーブルにトランザクションデータまたは測定データが保存され、1つ以上の小さいディメンションテーブルにデータ属性が保存されます。この構造は、[データウェアハウス](#)やビジネスインテリジェンスを用途とするように設計されています。

strangler fig パターン

レガシーシステムが廃止されるまで、システム機能を段階的に書き換えて置き換えることにより、モノリシックシステムをモダナイズするアプローチ。このパターンは、宿主の樹木から根を成長させ、最終的にその宿主を包み込み、宿主に取って代わるイチジクのつるを例えています。そのパターンは、モノリシックシステムを書き換えるときのリスクを管理する方法として [Martin Fowler](#) により提唱されました。このパターンの適用方法の例については、「[コンテナと Amazon API Gateway を使用して、従来の Microsoft ASP.NET \(ASMX\) ウェブサービスを段階的にモダナイズ](#)」を参照してください。

サブネット

VPC 内の IP アドレスの範囲。サブネットは、1つのアベイラビリティゾーンに存在する必要があります。

監視制御とデータ取得 (SCADA)

製造分野において、ハードウェアとソフトウェアを使用して物理アセットと本番運用をモニタリングするシステム。

対称暗号化

データの暗号化と復号に同じキーを使用する暗号化のアルゴリズム。

合成テスト

ユーザーとのやり取りをシミュレートして、起こり得る問題を検出したり、パフォーマンスをモニタリングしたりすることで、システムをテストします。[Amazon CloudWatch Synthetics](#) を使用すると、こうしたテストを作成できます。

システムプロンプト

コンテキスト、指示、ガイドラインなどを提示して、[LLM](#) に動作を指示する手法。システムプロンプトは、コンテキストを設定して、ユーザーとやり取りするルールを確立するのに有用です。

T

タグ

AWS リソースを整理するためのメタデータとして機能するキーと値のペア。タグは、リソースの管理、識別、整理、検索、フィルタリングに役立ちます。詳細については、「[AWS リソースのタグ付け](#)」を参照してください。

ターゲット変数

監督された機械学習でお客様が予測しようとしている値。これは、結果変数のことも指します。例えば、製造設定では、ターゲット変数が製品の欠陥である可能性があります。

タスクリスト

ランブックの進行状況を追跡するために使用されるツール。タスクリストには、ランブックの概要と完了する必要がある一般的なタスクのリストが含まれています。各一般的なタスクには、推定所要時間、所有者、進捗状況が含まれています。

テスト環境

「[環境](#)」を参照してください。

トレーニング

お客様の機械学習モデルに学習するデータを提供すること。トレーニングデータには正しい答えが含まれている必要があります。学習アルゴリズムは入力データ属性をターゲット (お客様が予測したい答え) にマッピングするトレーニングデータのパターンを検出します。これらのパターンをキャプチャする機械学習モデルを出力します。そして、お客様が機械学習モデルを使用して、ターゲットがわからない新しいデータでターゲットを予測できます。

トランジットゲートウェイ

VPC とオンプレミスネットワークを相互接続するために使用できる、ネットワークの中継ハブ。詳細については、AWS Transit Gateway ドキュメントの「[トランジットゲートウェイとは](#)」を参照してください。

トランクベースのワークフロー

デベロッパーが機能ブランチで機能をローカルにビルドしてテストし、その変更をメインブランチにマージするアプローチ。メインブランチはその後、開発環境、本番前環境、本番環境に合わせて順次構築されます。

信頼されたアクセス

ユーザーに代わって AWS Organizations およびそのアカウントで組織内でタスクを実行するために指定したサービスにアクセス許可を付与します。信頼されたサービスは、サービスにリンクされたロールを必要とときに各アカウントに作成し、ユーザーに代わって管理タスクを実行します。詳細については、ドキュメントの「[他の AWS のサービス AWS Organizations で使用する AWS Organizations](#)」を参照してください。

チューニング

機械学習モデルの精度を向上させるために、お客様のトレーニングプロセスの側面を変更する。例えば、お客様が機械学習モデルをトレーニングするには、ラベル付けセットを生成し、ラベルを追加します。これらのステップを、異なる設定で複数回繰り返して、モデルを最適化します。

ツーピザチーム

2 枚のピザを分け合えることができるくらい小さな DevOps チーム。ツーピザチームの規模では、ソフトウェア開発におけるコラボレーションに最適な機会が確保されます。

U

不確実性

予測機械学習モデルの信頼性を損なう可能性がある、不正確、不完全、または未知の情報を指す概念。不確実性には、次の 2 つのタイプがあります。認識論的不確実性は、限られた、不完全なデータによって引き起こされ、弁論的不確実性は、データに固有のノイズとランダム性によって引き起こされます。詳細については、[深層学習システムにおける不確実性の定量化ガイド](#)を参照してください。

未分化なタスク

ヘビーリフティングとも呼ばれ、アプリケーションの作成と運用には必要だが、エンドユーザーに直接的な価値をもたらさなかったり、競争上の優位性をもたらしたりしない作業です。未分化なタスクの例としては、調達、メンテナンス、キャパシティプランニングなどがあります。

上位環境

「[環境](#)」を参照してください。

V

バキューミング

ストレージを再利用してパフォーマンスを向上させるために、増分更新後にクリーンアップを行うデータベースのメンテナンス操作。

バージョンコントロール

リポジトリ内のソースコードへの変更など、変更を追跡するプロセスとツール。

VPC ピアリング

プライベート IP アドレスを使用してトラフィックをルーティングできる、2 つの VPC 間の接続。詳細については、Amazon VPC ドキュメントの「[VPC ピア機能とは](#)」を参照してください。

脆弱性

システムのセキュリティを脅かすソフトウェアまたはハードウェアの欠陥。

W

ウォームキャッシュ

頻繁にアクセスされる最新の関連データを含むバッファキャッシュ。データベースインスタンスはバッファキャッシュから、メインメモリまたはディスクからよりも短い時間で読み取りを行うことができます。

ウォームデータ

アクセス頻度の低いデータ。この種類のデータをクエリする場合、通常は適度に遅いクエリでも問題ありません。

ウィンドウ関数

現在のレコードに何らかの形で関連している行のグループに計算を実行する SQL 関数。ウィンドウ関数は、移動平均を計算したり、現在の行の相対位置に基づいて他の行の値にアクセスするといったタスクの処理に役立ちます。

ワークロード

ビジネス価値をもたらすリソースとコード (顧客向けアプリケーションやバックエンドプロセスなど) の総称。

ワークストリーム

特定のタスクセットを担当する移行プロジェクト内の機能グループ。各ワークストリームは独立していますが、プロジェクト内の他のワークストリームをサポートしています。たとえば、ポートフォリオワークストリームは、アプリケーションの優先順位付け、ウェーブ計画、および移行メタデータの収集を担当します。ポートフォリオワークストリームは、これらの設備を移行ワークストリームで実現し、サーバーとアプリケーションを移行します。

WORM

「[Write-Once-Read-Many](#)」を参照してください。

WQF

「[AWS ワークロード資格フレームワーク](#)」を参照してください

Write-Once-Read-Many (WORM)

データを 1 回のみ書き込むことで、データの削除や変更を防ぐストレージモデル。承認済みユーザーは、必要な回数だけデータを読み取ることができますが、変更することはできません。このデータストレージインフラストラクチャは、[イミュータブル](#)と見なされます。

Z

ゼロデイ 익스プロイト

[ゼロデイ脆弱性](#)を悪用した攻撃 (一般的にマルウェアによる)。

ゼロデイ脆弱性

実稼働システムにおける未解決の欠陥または脆弱性。脅威アクターは、このような脆弱性を利用してシステムを攻撃する可能性があります。開発者は、よく攻撃の結果で脆弱性に気付きます。

ゼロショットプロンプト

[LLM](#) にタスク実行の手順は提示するが、実行のガイドとして役立つ例 (ショット) は提示しない方法。LLM は、事前トレーニング済みの知識を使用してタスクを処理する必要があります。ゼロショットプロンプトの有効性は、タスクの複雑さとプロンプトの品質によって異なります。「[数ショットプロンプト](#)」も参照してください。

ゾンビアプリケーション

平均 CPU およびメモリ使用率が 5% 未満のアプリケーション。移行プロジェクトでは、これらのアプリケーションを廃止するのが一般的です。

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。