マルチアカウント DevOps 環境の Git 分岐戦略の選択

AWS 規範ガイダンス

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

Table of Contents

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。互換性のある Git 分岐戦略を選択して your DevOps プロセスと組み合わせることは、成功に不可欠です。組織に適した Git 分岐戦略を選択することで、開発チーム間で DevOps 標準とベストプラクティスを簡潔に伝達できます。Git の分岐は 1 つの環境で簡単に行えますが、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。環境が複数あると、 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示します。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。

に関する DevOps のベストプラクティスについては AWS、 AWS Well-Architected のDevOps Guidance」を参照してください。このガイドを読む際には、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てください。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 を持つ組織の DevOps 分岐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、要件、目標、ベストプラクティスを最初から最もよく満たす戦略を適用して、 でのエクスペリエンスを効率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 GitHub 環境の a DevOps 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、他の組織やチーム、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、継続的インテグレーションと継続的デリバリーを実装することを推奨しています (CI/CD), which is the process of automating the software release lifecycle. It automates much or all of the manual DevOps processes that are traditionally required to get new code from development into production. A CI/CD pipeline encompasses the sandbox, development, testing, staging, and production environments. In each environment, the CI/CD pipeline provisions any infrastructure that is needed to deploy or test the code. By using CI/CD, development teams can make changes to code that are then automatically tested and deployed. CI/CD パイプラインは、開発チームに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一貫性、標準、ベストプラクティス、最小承認レベルを適用します。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分岐戦略はすべて、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 が開発チームのガードレールとメカニズムを確立し、デベロッパー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ようにします。

AWS は、CI/CD パイプラインの構築に役立つように設計されたデベロッパーサービスのスイートを提供します。例えば、 AWS CodePipelineは、リリースパイプラインを自動化して、迅速かつ信頼性の高いアプリケーションとインフラストラクチャの更新を行うのに役立つフルマネージドの継続的デリバリーサービスです。 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、「 のデベロッパーツール AWS」を参照してください。

環境を理解する DevOps

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段階的に分け、それらのアクティビティをモニタリングし、未承認の機能の意図しないリリースを防ぐことができます。環境ごとに 1 つ以上の AWS アカウント を持つことができます。

ほとんどの組織では、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、異なる名前で呼び出される可能性があります。

- サンドボックス — デベロッパーがコードを記述し、ミスをし、概念実証作業を実行する環境。
- 開発 — デベロッパー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する本番稼働前の環境。この環境は、本番環境と可能な限り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、次の環境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
- テスト環境

- 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では、デベロッパーがコードを記述し、ミスをし、概念実証作業を実行します。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、ソースコードを構築してコンパイルする
4. ユニットテストの実行
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の抽出
10. ビルドアーティファクトの発行

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 フローモデルを使用して

いる場合は、次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ニングを実行する
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`ブランチが正常に構築されました
- 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、リリースする準備ができています。 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さ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、タスクを実行するために最低限必要な権限を付与する際の、セキュリティのベストプラクティスです。デベロッパーは、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、ソースコードを構築してコンパイルする
4. ユニットテストの実行

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. IaC の構築
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の抽出

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に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ニングを実行する
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

- 開発環境で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が成功する
- ユニットテストが 100% で合格
- IaC 構築の成功
- デプロイアーティファクトが正常に作成された
- デベロッパー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

テスト環境

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、ソースコードを構築してコンパイルする
4. ユニットテストの実行
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. IaC の構築
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の抽出

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ニングを実行する
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の実行

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ています。
- 開発チームは、検出されたバグをbugfixブランチを通じて解決しました。

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。例えば、データ設定の範囲とサイズは本番ワークロードと似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ンストレーション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。デベロッパーは、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ニングを実行する
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) ロードテストを実行する
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

本稼働環境は、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、最小特権によって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 で読み取り専用アクセスを制限する必要があります。例えば、デベロッパーは day-to-day オペレーション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、本番環境に`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。本番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ニングを実行する
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 も参照してください。以下は、 での Git ベースの開発の主要なベストプラクティスです AWS。

- 変更を小規模かつ頻繁に保つ — 開発者に、小規模で増分的な変更や機能をコミットするよう促します。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 – 不完全な機能や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、または無効にすることができます。
- 堅牢なテストスイートの維持 — 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。テスト自動化に投資し、失敗したテストの修正を優先します。
- 継続的な統合の採用 — 継続的な統合ツールとプラクティスを使用して、コードの変更を自動的に構築、テスト、ブランチ `develop` (Gitflow) または`main`ブランチ (トランクまたは GitHub フロー) に統合します。これにより、問題を早期に発見し、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 – ビルドが中断または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) が解放可能な状態に保たれ、他のデベロッパーへの影響が最小限に抑えられます。
- コミュニケーションとコラボレーション — チームメンバー間でオープンなコミュニケーションとコラボレーションを推進します。デベロッパーが進行中の作業や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にリファクタリングして、保守性を向上させ、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。
- 複雑なタスクに有効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、有効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の導入に関する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、Git ベースの以下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 trunkまたは ブランチと呼ばれる単一のmainブランチで作業するソフトウェア開発のプラクティスです。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頼ることで、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 フロー – GitHub フローは、 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。これは、存続期間の短いfeatureブランチという考え方に基づいています。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、releaseブランチが作成され、その機能が上位環境にデプロイされます。

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、各分岐戦略を詳細に確認し、そのニュアンスを理解し、組織のユースケースに適合す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 trunkまたは ブランチと呼ばれる単一のmainブランチで作業するソフトウェア開発プラクティスです。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頼ることで、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、デベロッパーは 1 日に複数回mainブランチに変更をコミットし、小規模な増分更新を目指します。これにより、迅速なフィードバックループが可能になり、マージの競

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が強調されています。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に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。

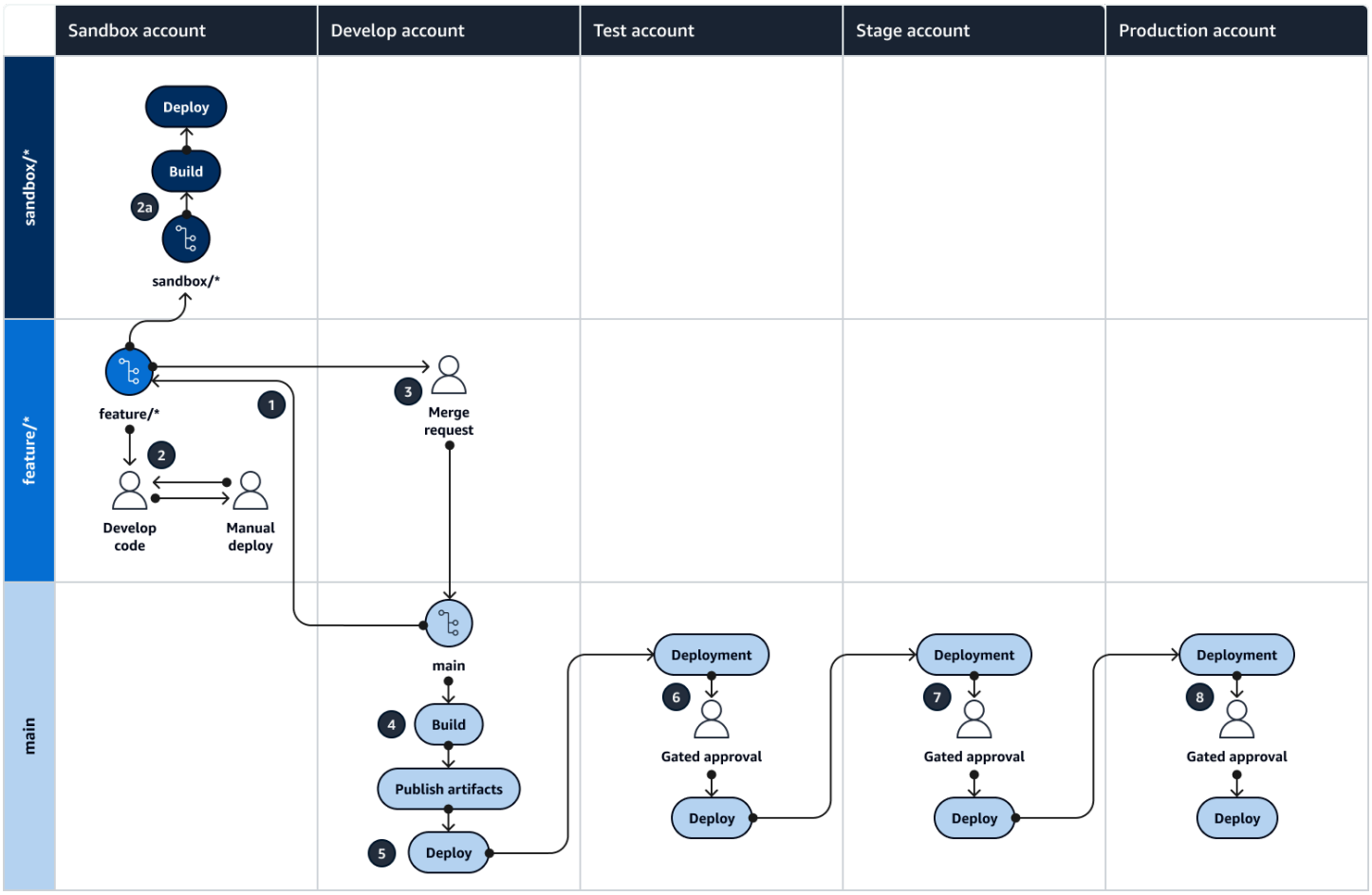
- マルチアカウント DevOps 環境のトランク分岐戦略を実装する (AWS 規範ガイダンス)
- トランクベースの開発入門 (トランクベースの開発ウェブサイト)

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

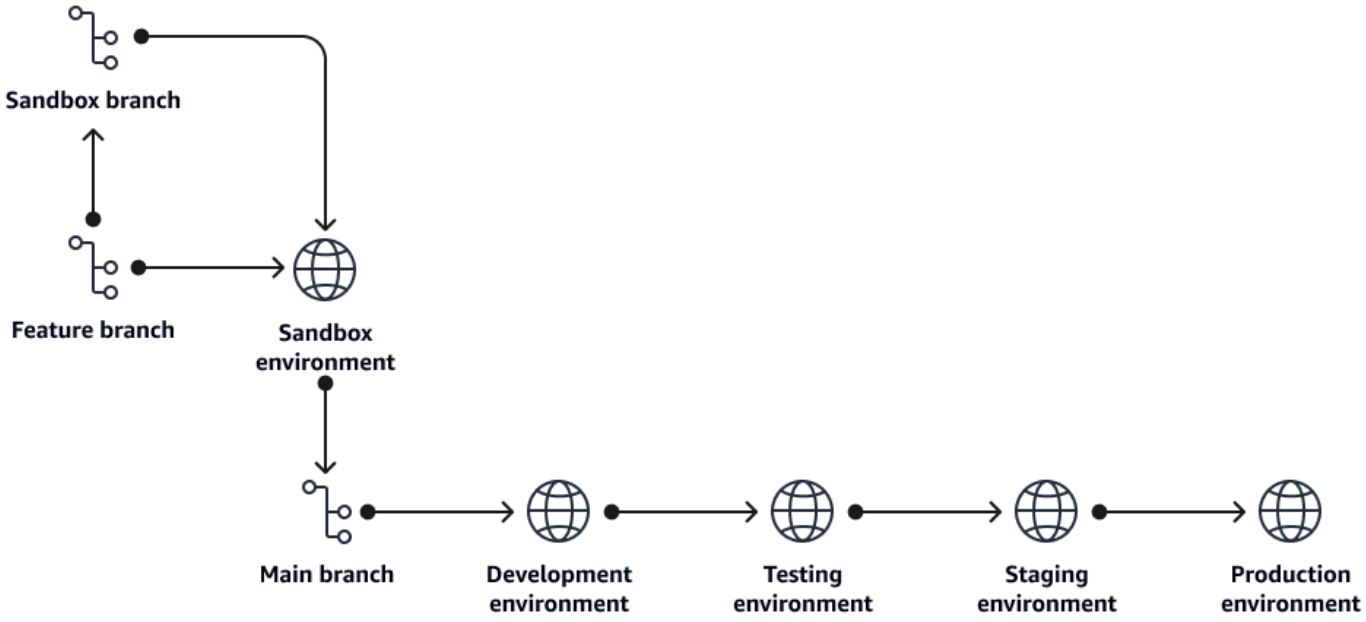
トランク戦略の視覚的な概要

次の図は、Punnett 四角形 (Wikipedia) のように使用して、トランク分岐戦略を理解できます。垂直軸のブランチを水平軸の AWS 環境に合わせて並べ、各シナリオで実行するアクションを決定します。丸で囲まれた数字は、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の、トランクmain分岐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。



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、修正プログラムを作成します。feature ブランチを作成するには、ブランチからmain分岐します。デベロッパーは、featureブランチ内のコードを反復、コミット、テストします。機能が完了すると、デベロッパーはその機能を昇格します。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のトランクブランチですが、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で使用されます。

- 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、短期間の使用を目的としています。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、開発されてから 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することができます。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、コミュニケーションスキルが強い、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。大規模な開発チームやフラグメント化された開発チームがある場合や、スケジュールされた機能リリースを再現し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では、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、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。
- マージの競合の軽減 – 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の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できるようになり、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よりまとまりやすくなります。
- 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、徹底的なコードレビューの実行が容易になります。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 — 統合、テスト、コードレビューが頻繁に行われるため、トランクベースの開発は全体的なコード品質の向上に役立ちます。デベロッパーは問題をより迅速に検出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、開発プロセスや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、いくつかの顕著な欠点です。

- 分離の制限 – すべてのデベロッパーが同じブランチで作業するため、その変更はチーム内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

ドが中断されたりする可能性があります。対照的に、機能ベースの開発では、開発者がより独立して作業できるように、変更をより適切に分離します。

- 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、継続的な統合と自動テストによって問題を迅速に検出できます。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、適切に保守されたテストスイートが必要です。テストが包括的でないか信頼性が低い場合、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。これは有利な場合がありますが、厳密なリリーススケジュールを持つプロジェクトや、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。機能ベースの開発により、機能のリリース時期とリリース方法をより詳細に制御できます。
- コードチャーン — デベロッパーが変更をメインブランチに常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、デベロッパー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度な統制、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デベロッパーと作業する場合、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、ビルドブレー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、 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。 GitHub フローは、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有効期間の短い機能ブランチの考え方に基づいています。 GitHub フロー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特徴量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コラボレーション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。
- マージリクエスト – デベロッパーはマージリクエストを使用して、変更をメインブランチにマージする前にディスカッションとレビュープロセスを開始します。

GitHub フローの詳細については、次のリソースを参照してください。

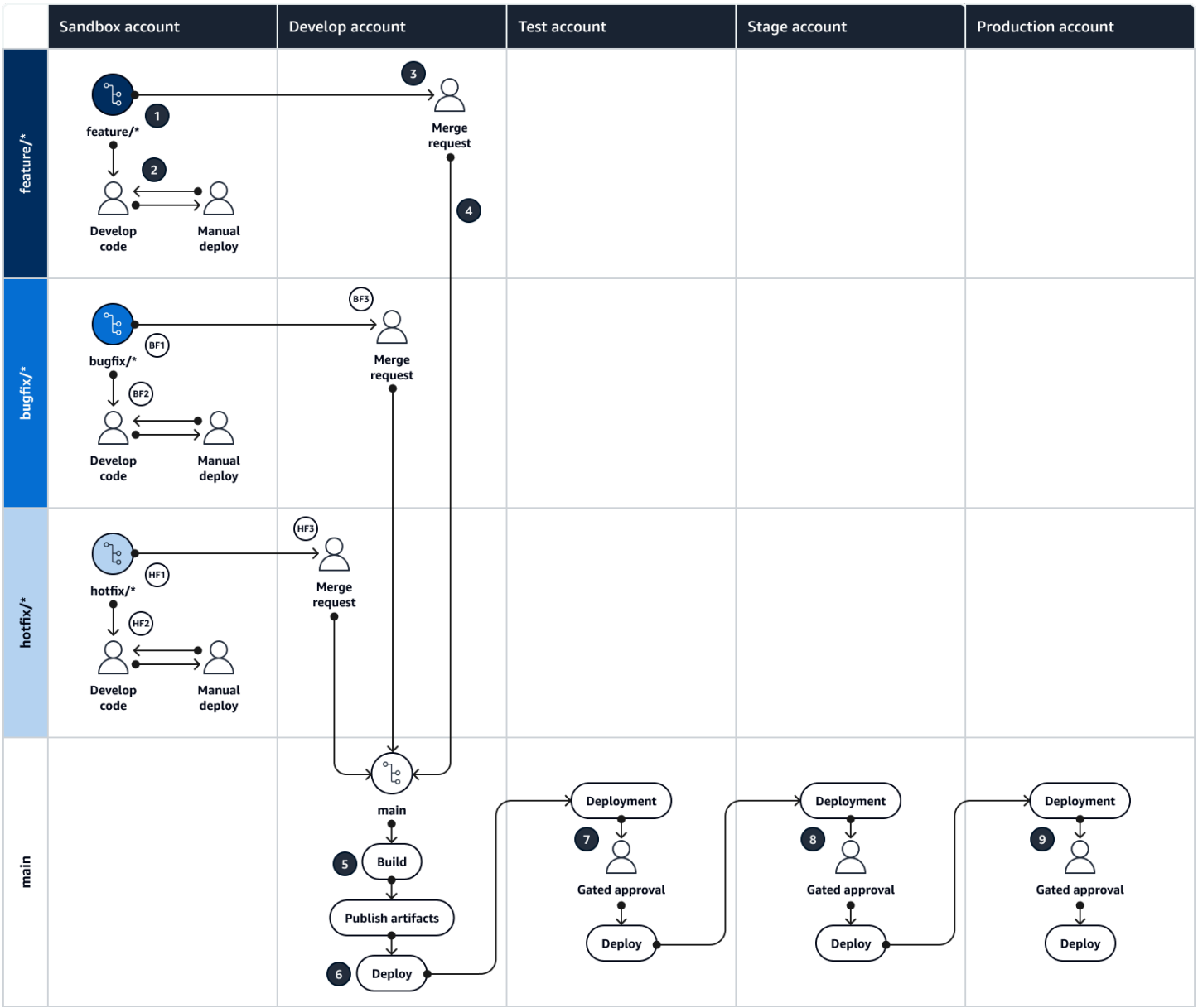
- マルチアカウント DevOps 環境の GitHub フロー分岐戦略を実装する (AWS 規範ガイダンス)
- GitHub フロークイックスタート (GitHub ドキュメント)
- GitHub フローの理由 (GitHubフローウェブサイト)

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

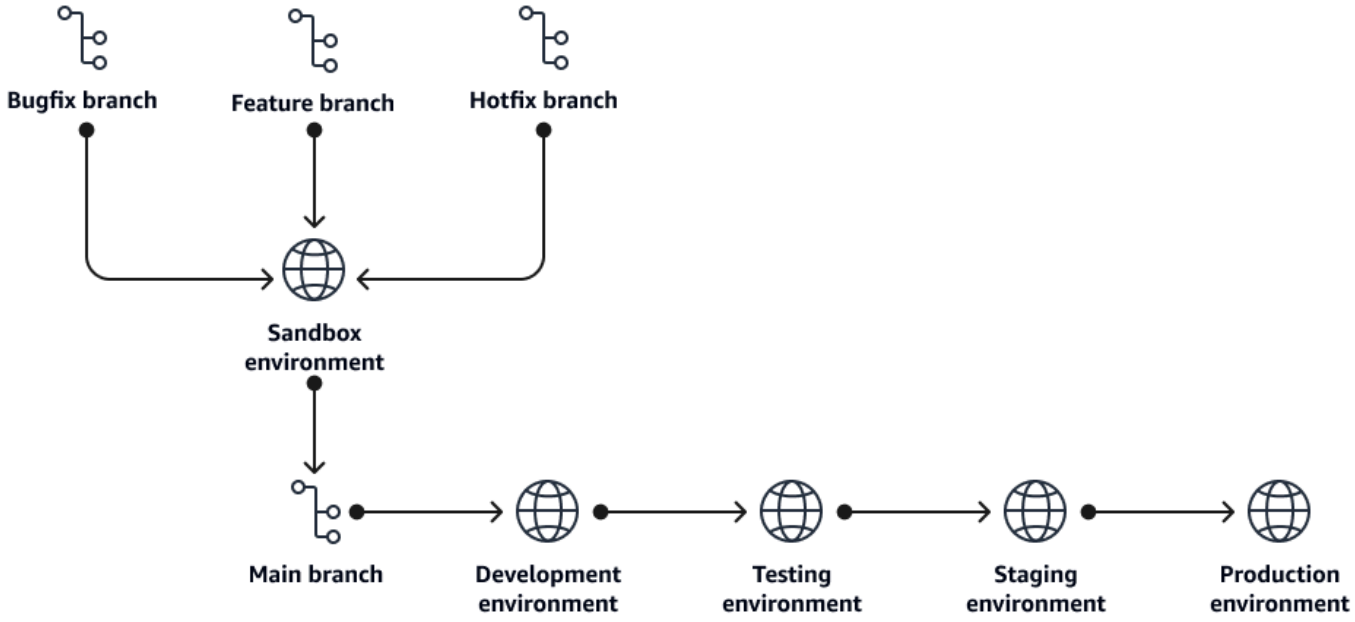
GitHub フロー戦略の視覚的な概要

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。丸で囲まれた数字は、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 フロー分岐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、ブランチからmain分岐します。デベロッパーは、featureブランチ内のコードを反復、コミット、テストします。機能が完了すると、デベロッパーはマージリクエストを作成して機能を昇格しますmain。

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ブランチからmain分岐されます。バグ修正がサンドボックスまたは下位環境のいずれかでテストされた後、マージリクエストmainを通じて にマージすることで、上位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

この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最小限の遅延で、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main分岐されます。ホットフィックスをサンドボックスまたは下位環境のいずれかでテストした後、マージリクエストmainを通じて にマージすることで、上位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、コミュニケーションスキルが高い、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、継続的なデリバリーを実装したいチームや、一般的な CI/CD エンジンで十分にサポートされているチームに適しています。 GitHub フローは軽量で、ルールがあまりなく、動きの速い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 フローには、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 フローは、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、複雑さを最小限に抑えながら、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 フローは、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集中的な変更が奨励され、効率が向上します。
- バージョン管理のクリア — GitHub フローでは、すべての変更が個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、デベロッパー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 フロー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、自動テストとデプロイプロセスを開始できます。CI ツールは、`main`ブランチにマージされる前に変更を徹底的にテストし、コードベースにバグを導入するリスクを軽減するのに役立ちます。
- 迅速なフィードバックと継続的な改善 — GitHub フローは、プルリクエストを通じてコードレビューや議論を頻繁に促進することで、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、最終的には開発チーム内でのコード品質が向上し、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 フローは、特に Git とバージョン管理の概念に慣れているチームにとって、Gitflow よりも学習と導入が容易になります。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デベロッパー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軽減されます。
- 継続的な開発 — GitHub フローにより、チームは`main`、ブランチにマージされた直後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、継続的なデプロイアプローチを採用できます。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 – 複雑なコードベースと複数の長期的な特徴ブランチを持つ大規模なプロジェクトには、 GitHub フローが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 では、リリースプロセスや、バージョニング、修正、メンテナンスブランチなどのサポート機能は明示的に定義されていません。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、長期サポートとメンテナンスを必要とするプロジェクトでは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、存続期間の短い特徴量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特徴量の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 フローの制約内では難しい場合があります。
- マージの競合が頻繁に発生する可能性がある – GitHub フロー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の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。
- 形式化されたワークフローフェーズの欠如 – GitHub フローでは、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は定義されません。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 フローは`main`頻繁にブランチへの変更のマージを促すため、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コードを開発から本番稼働に移行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、個々の特徴量ブランチで完了し、承認されて、統合に使用される開発ブランチにマージされます。このブランチの機能は、本番稼働可能と見なされます。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、どの変更がどの名

前付き環境に移動するか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。

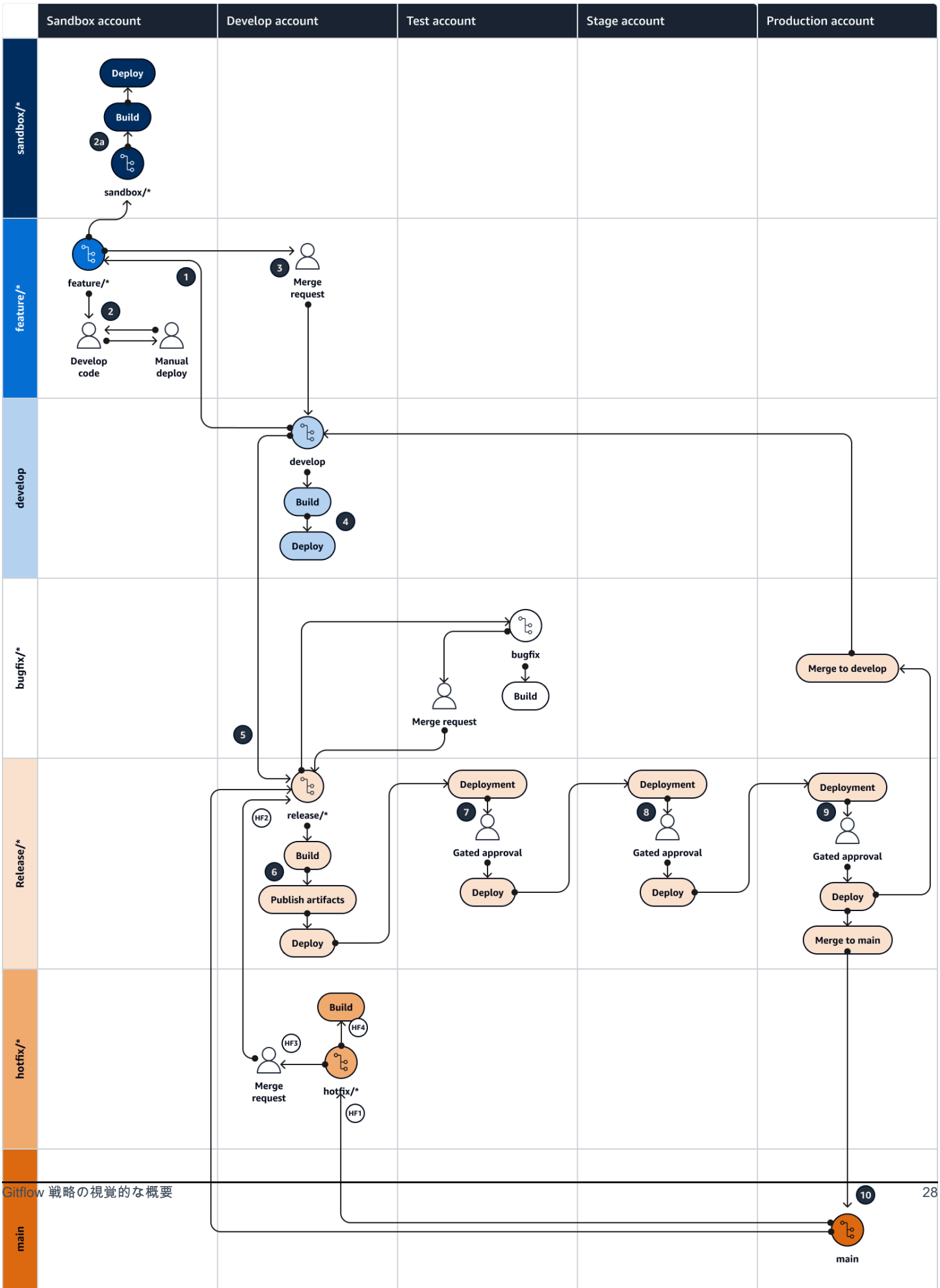
- マルチアカウント DevOps 環境の Gitflow 分岐戦略を実装する (AWS 規範ガイダンス)
- 元の Gitflow ブログ (Vincent Driessen ブログ記事)
- Gitflow ワークフロー (アトラシアン)

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

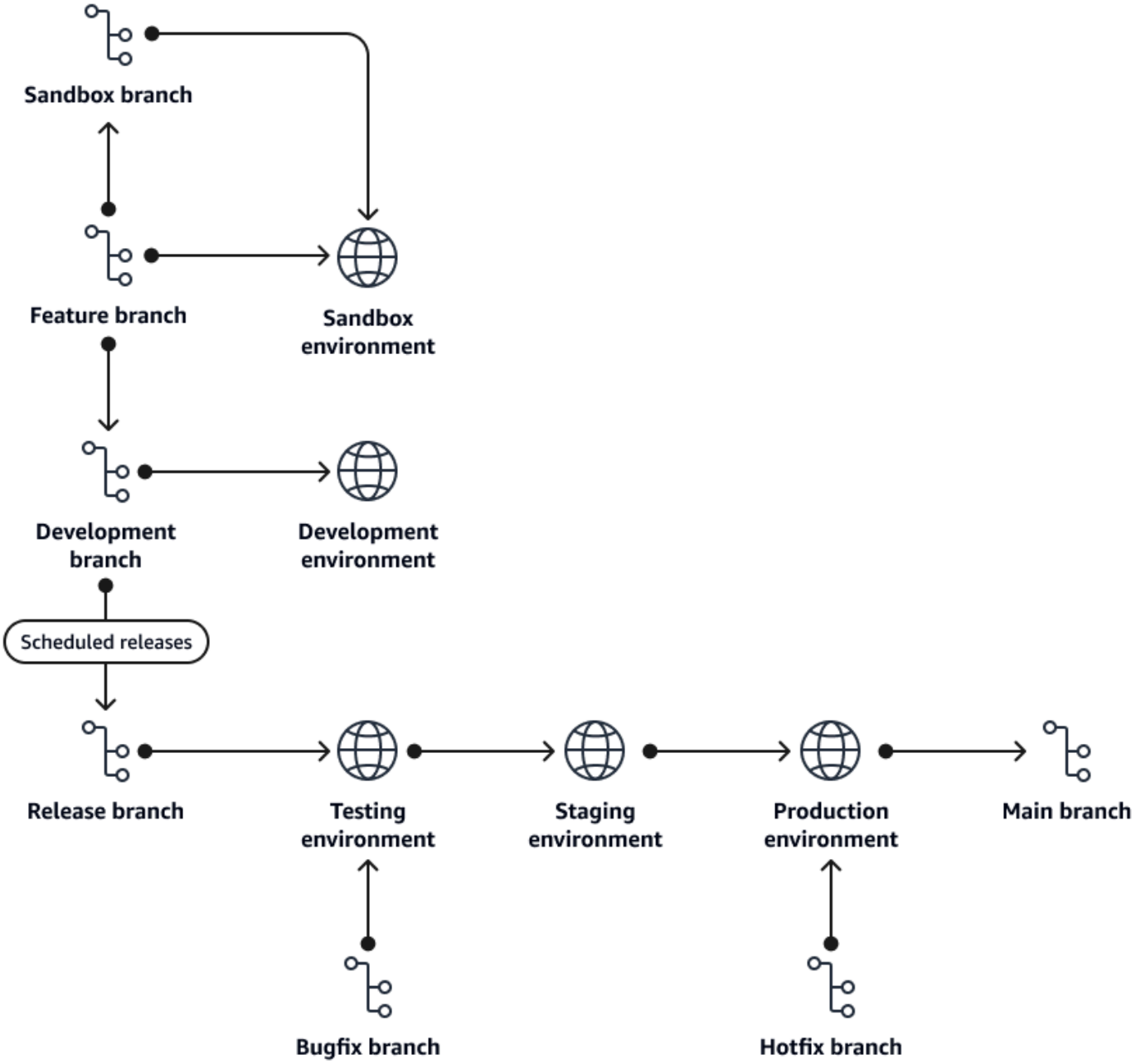
次の図は、Gitflow 分岐戦略を理解するために Punnett の四角形のように使用できます。垂直軸のブランチを水平軸の AWS 環境に合わせて並べ、各シナリオで実行するアクションを決定します。丸で囲まれた数字は、図に示されている一連のアクションをガイドします。各環境で発生するアクティビティの詳細については、このガイドの「 DevOps 環境」を参照してください。



Gitflow 戦略の視覚的な概要

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。デベロッパーは、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、デベロッパー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 の非標準の短期ブランチです。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で使用されます。

- 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 ブランチは本質的に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_Pipeline_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 つのデベロッパー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、ビルドワンス、デプロイ、または多く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、ステージング、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、デプロイを続行す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、バグ修正や修正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 ブランチは存続期間の長いブランチで、常に本番環境で実行されているコードを表します。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ブランチmain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、ステージング、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 ブランチは常にブランチからrelease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的なブランチです。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場合にのみ使用されます。hotfix ブランチは常に から分岐されますmain。

修正がテストされたら、 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、修正を本番環境に昇格させることができます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型の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、多くの場合、これは大規模な組織で優先されます。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、戦略に組み込まれているレビューと品質保証の機会が複数あるためです。Gitflow は、本番リリースの複数のバージョン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、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、継続的な配信を試みる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が長いブランチであり、適切な方法で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、定期的な開発とリリースのケイデンスを持つチームに適しています。
- コラボレーションの向上 – Gitflow では、 `feature`ブランチと `release`ブランチの使用を推奨しています。これら 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`release`ブランチを使用します。ブランチは存続期間が長いブランチである可能性があります。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。
- コード品質レビューの組み込み — 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。このプロセスにより、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、デベロッパー間の摩擦がなくなります。
- 組織のメリット — Gitflow には組織レベルでも利点があります。Gitflow では、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ています。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、開発プロセスやチームのダイナミクスに影響を与える可能性のある欠点がいくつかあります。以下は、いくつかの顕著な欠点です。

- 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。
- 継続的なデプロイ — 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 では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が必要であるためです。
- ブランチ管理 – Gitflow は多くのブランチを使用しますが、維持するのは面倒な作業になる可能性があります。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 のリリースは通常、他の分岐モデルよりも遅いため、リリース用に蓄積される特徴量が増える可能性があり、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は、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 規範ガイダンス

- 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 ガイダンス

- AWS DevOps ガイダンス
- 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 規範的ガイダンスの用語集

以下は、 AWS 規範的ガイダンスが提供する戦略、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します。
- 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にある程度の最適化を導入します。例: オンプレミスの Oracle データベースを の Oracle 用 Amazon Relational Database Service (Amazon RDS) に移行します AWS クラウド。
- 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 に移行します。
- 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: オンプレミスの Oracle データベースを の EC2 インスタンス上の Oracle に移行します AWS クラウド。
- 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。サーバーを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移行します。例: の移行 Microsoft Hyper-V アプリケーション 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、およびそれらを移行す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。
- 使用停止 — 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コントロール」を参照してください。

抽象化されたサービス

「 マネージドサービス」を参照してください。

ACID

原子性、一貫性、分離性、耐久性を参照してください。

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ですが、より多くの作業が必要です。

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の方法。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

行のグループに対して動作し、グループの単一の戻り値を計算する SQL 関数。集計関数の例には、 SUMや などがありますMAX。

AI

人工知能を参照してください。

AIOps

「人工知能オペレーション」を参照してください。

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。

アプリケーションコントロール

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 の需要要素であり、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?」を参照してください。

人工知能オペレーション (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。AIOps が移行戦略で AWS どのように使用されるかの詳細については、「 オペレーション統合ガイド」を参照してください。

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。

アトミック性、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コントロール (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 （ ABAC) ドキュメントの「 の AWS IAM」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。

アベイラビリティーゾーン

他のアベイラビリティーゾーンの障害から 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ー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 内の個別の場所。

AWS クラウド導入フレームワーク (AWS CAF)

組織がクラウドに正常に移行するための効率的で効果的な計画を立て AWS るのに役立つ、 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分類します。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に関する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 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

個人または組織に混乱や損害を与えることを目的としたボット。

BCP

事業継続計画を参照してください。

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 で動作グラフを使用して、失敗したログオン試行、疑わしい API 呼び出し、および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Data in a behavior graph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。エンディアンネスも参照してください。

二項分類

バイナリ結果 (2 つの可能なクラスのうちの 1 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

2 つの異なる同一の環境を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は 1 つの環境 (青) で実行し、新しいアプリケーションバージョンは他の環境 (緑) で実行します。この戦略は、影響を最小限に抑えながら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インタラクション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ーラーなど、有益または有益なボットもあります。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ているものがあります。

ボットネット

マルウェアに感染し、ボットのヘルダーまたはボットオペレーターとして知られる 1 人の当事者による管理下にあるボットのネットワーク。ボットは、ボットとその影響をスケールするための最もよく知られているメカニズムです。

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、「ブランチについて」(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なります。詳細については、 Well-Architected ガイダンスの「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。 AWS

ブラウンフィールド戦略

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ン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。

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、ホワイトペーパー AWSでのコンテナ化されたマイクロサービスの実行 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。

事業継続計画 (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。

Canary デプロイ

エンドユーザーへのバージョンの低速かつ段階的なリリース。自信が持てたら、新しいバージョンをデプロイし、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。

データキャプチャの変更 (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、同期を維持するためにターゲットシステムの変更を監査またはレプリケートするなど、さまざまな目的で使用できます。

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、システムの回復力をテストします。AWS Fault Injection Service（ AWS FIS ）を使用して、 AWS ワークロードに負荷を与え、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

ターゲットがデータ AWS のサービス を受信する前に、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テクノロジー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 に移行するときに通常実行する 4 つのフェーズ AWS クラウド：

- 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
- 基盤 — クラウド導入を拡大するための基本的な投資 (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。これらが AWS 移行戦略とどのように関連しているかについては、「移行準備ガイド」を参照してください。

CMDB

「設定管理データベース」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、次のようなものがあります。GitHub または Bitbucket Cloud。 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、デジタル画像や動画などのビジュアル形式から情報を分析および抽出する AI の分野。例えば、 はオンプレミスのカメラネットワークに CV を追加するデバイス AWS Panorama を提供し、Amazon SageMaker は CV の画像処理アルゴリズムを提供します。

設定ドリフト

ワークロードの場合、設定は想定した状態から変化します。これにより、ワークロードが非準拠になる可能性があり、通常は段階的かつ意図的ではありません。

設定管理データベース (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、移行のポートフォリオ検出および分析段階で CMDB のデータを使用します。

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 とリージョン、または組織全体に 1 つのエンティティとしてデプロイできます。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 is commonly described as a pipeline. CI/CD は、プロセスの自動化、生産性の向上、コード品質の向上、より迅速な提供に役立ちます。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

一元的な管理とガバナンスにより、分散型の分散データ所有権を提供す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していることを確実にします。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通常、大量の履歴データが含まれており、クエリや分析によく使用されます。

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る。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。

ディープラーニング

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。

defense-in-depth

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。例えば、a defense-in-depth 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で使用できるサービスを参照してください。

デプロイメント

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、Implementing security controls on AWSのDetective controlsを参照してください。

開発値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、もともとリーンな製造プラクティス向け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ます。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

スタースキーマでは、ファクトテーブル内の量的データに関するデータ属性を含む小さなテーブル。ディメンションテーブル属性は通常、テキストフィールドまたはテキストのように動作する

離散数値です。これらの属性は、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よく使用されます。

ディザスタ

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)

災害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に使用する戦略とプロセス。詳細については、 AWS Well-Architected フレームワークの「 でのワークロードのディザスタリカバリ AWS: クラウドでのリカバリ」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 (ボストン: 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については、「コンテナと Amazon Word API Gateway を使用してレガシー Microsoft ASP.NET (ASMX) ウェブサービスを段階的にモダナイズする」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逸脱の追跡。例えば、 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 ガバナンス要件のコンプライアンスに影響を与える可能性のあるランディングゾーンの変更を検出したりできます。

DVSM

「開発値ストリームマッピング」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、応答時間を短縮できます。

電子データ交換 (EDI)

組織間の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プロセス。

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

Virtual Private Cloud (VPC) 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 Word) ドキュメントの「エンドポイントサービスの作成」を参照してください。 VPC

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使用できる、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、サマリー統計を計算し、データの視覚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

星スキーマの中央テーブル。事業運営に関する量的データを保存します。通常、ファクトテーブルには、メジャーを含む列とディメンションテーブルへの外部キーを含む列の 2 種類の列が含まれます。

フェイルファスト

頻繁で段階的なテストを使用して開発ライフサイクルを短縮する哲学。これはアジャイルアプローチの重要な部分です。

障害分離の境界

では AWS クラウド、アベイラビリティーゾーン、コントロールプレーン AWS リージョン、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の向上に役立ちます。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 Explanations (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として表されます。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

同様のタスクを実行するように求める前に、タスクと必要な出力を示す少数の例を LLM に提供します。この手法は、プロンプトに埋め込まれた例 (ショット) からモデルが学習するコンテキスト内学習のアプリケーションです。少数ショットプロンプトは、特定のフォーマット、推論、またはドメイン知識を必要とする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コントロール」を参照してください。

きめ細かなアクセスコントロール (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

段階的なアプローチを使用するのではなく、変更データキャプチャによる継続的なデータレプリケーションを使用して、可能な限り短い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)

一般化データとラベルなしデータの大規模なデータセットでトレーニングされている大規模な深層学習ニューラルネットワーク。FMsは、言語の理解、テキストと画像の生成、自然言語での会話など、さまざまな一般的なタスクを実行できます。詳細については、「基礎モデルとは」を参照してください。

G

生成 AI

大量のデータに対してトレーニングされ、シンプルなテキストプロンプトを使用してイメージ、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る AI モデルのサブセット。詳細については、「生成 AI とは」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)

Amazon CloudFront では、特定の国のユーザーがコンテンツディストリビューションにアクセスできないようにするオプションです。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分散の制限」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 ワークフローはレガシーと見なされ、トランクベースのワークフローはモダンで推奨されるアプローチです。

ゴールデンイメージ

システムまたはソフトウェアの新しいインスタンスをデプロイするためのテンプレートとして使用されるシステムまたはソフトウェアのスナップショット。例えば、製造では、ゴールデンイメージを使用して複数のデバイスにソフトウェアをプロビジョニングし、デバイスの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 (OUs) 全体のリソース、ポリシー、コンプライアンスの管理に役立つ大まかなルール。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、、、Amazon GuardDuty AWS Security Hub、、 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。

ハイアベイラビリティ (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に使用されるデータセットから保留されている、ラベル付きの履歴データの一部。ホールドアウトデータを使用してモデル予測をホールドアウトデータと比較することで、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

ソースデータベースを、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) に移行します。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。緊急性のため、通常、ホットフィックスは一般的な DevOps リリースワークフローの外部で行われます。

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

IaC

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CPUおよびメモリ使用量が 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

「産業モノのインターネット」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

既存のインフラストラクチャを更新、パッチ適用、または変更するのではなく、本番環境のワークロードに新しいインフラストラクチャをデプロイするモデル。イミュータブルなインフラストラクチャは、本質的にミュータブルなインフラストラクチャよりも一貫性、信頼性、予測性が高くなります。詳細については、 AWS 「 Well-Architected フレームワーク」の「イミュータブルインフラストラクチャを使用したデプロイ」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 (イングレス) VPC

AWS マルチアカウントアーキテクチャでは、アプリケーション外からの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 セキュリティリファレンスアーキテクチャで

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VPCsを使用してネットワークアカウントを設定することをお勧めします。

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 によって導入された用語は、接続、リアルタイムデータ、自動化、分析、AI/ML の進歩によるビジネスプロセスのモダナイゼーションを指します。

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。

産業モノのインターネット (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、「産業モノのインターネット (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。

検査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 セキュリティリファレンスアーキテクチャでは、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VPCsを使用してネットワークアカウントを設定することをお勧めします。

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、AWS による機械学習モデルの解釈可能性」を参照してください。

IoT

「モノのインターネット」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

ラベルベースのアクセスコントロール (LBAC)

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ている必須アクセスコントロール (MAC) の実装。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)

大量のデータに基づいて事前トレーニングされた深層学習 AI モデル。LLM は、質問への回答、ドキュメントの要約、テキストの他の言語への翻訳、文の完了など、複数のタスクを実行できます。詳細については、LLMs とは」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

「ラベルベースのアクセスコントロール」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「最小特権の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 R」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。エンディアンネスも参照してください。

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またはプライバシーを侵害するように設計されているソフトウェア。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、不正アクセスにつながる可能性があります。マルウェアの例としては、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 が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、ユーザーが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する。Amazon Simple Storage Service (Amazon S3) と Amazon DynamoDB は、 マネージドサービスの例です。これら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、制御するためのソフトウェアシステム。これにより、生産現場の生産物が完成した製品に変換されます。

MAP

「移行促進プログラム」を参照してください。

メカニズム

ツールを作成し、ツールの採用を推進し、調整のために結果を検査する完全なプロセス。メカニズムは、動作中にそれ自体を強化して改善するサイクルです。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。

メッセージキューイングテレメトリトランスポート (MQTT)

リソースに制約のある IoT デバイス用の、パブリッシュ/サブスクライブパターンに基づく軽量な machine-to-machine (M2M) 通信プロトコル。

マイクロサービス

明確に定義された APIs を介して通信し、通常は小規模で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。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、軽量な APIs 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)

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。組織がクラウドへの移行のための強固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ちます。MAP には、従来の移行を体系的に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ための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と所有者、移行エンジニア、デベロッパー、スプリントに取り組む 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には、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 アプリケーション移行サービスを使用して Amazon EC2 への移行をリホストします。

移行ポートフォリオ評価 (MPA)

に移行するためのビジネスケースを検証するための情報を提供するオンラインツール AWS クラウド。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) と移行計画 (アプリケーションデータ分析とデータ収集、アプリケーショングループ化、移行の優先順位付け、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 移行戦略の最初のフェーズです。

移行戦略

ワークロードを に移行するために使用するアプローチ AWS クラウド。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、「」の「アプリケーションをモダナイズするための戦略 AWS クラウド」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、「」の「アプリケーションのモダナイゼーション準備状況の評価 AWS クラウド」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

「移行ポートフォリオ評価」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。

ミュータブルインフラストラクチャ

本番ワークロードの既存のインフラストラクチャを更新および変更するモデル。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をベストプラクティスとして使用することを推奨しています。

O

OAC

「オリジンアクセスコントロール」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

「組織の変更管理」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

「 オペレーションの統合」を参照してください。

OLA

「運用レベルの契約」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。

オープンプロセス通信 - 統合アーキテクチャ (OPC-UA)

産業用オートメーション用の A machine-to-machine (M2M) 通信プロトコル。OPC-UA は、データの暗号化、認証、認可スキームを備えた相互運用性標準を提供します。

運用レベルの契約 (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 IT グループが相互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)

インシデントや潜在的な障害の理解、評価、防止、または範囲の縮小に役立つ質問のチェックリストと関連するベストプラクティス。詳細については、 AWS Well-Architected フレームワークの「運用準備状況レビュー (ORR)」を参照してください。

運用テクノロジー (OT)

物理環境と連携して産業運用、機器、インフラストラクチャを制御するハードウェアおよびソフトウェアシステム。製造では、OT と情報技術 (IT) システムの統合が、Industry 4.0 トランスフォーメーションの主要な焦点です。

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 のすべてのイベントをログ AWS CloudTrail に記録する 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、変化の採用を加速し、移行に伴う問題に対処し、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や戦略に備え、移行するのに役立ちます。 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から、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。

オリジンアクセスコントロール (OAC)

In CloudFront は、Amazon Simple Storage Service (Amazon S3) コンテンツを保護するためのアクセスを制限するための拡張オプションです。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、および 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)

In CloudFront は、Amazon S3 コンテンツを保護するためのアクセスを制限するためのオプションです。OAI を使用すると、 CloudFront は Amazon S3 が認証できるプリンシパルを作成します。認証されたプリンシパルは、特定の CloudFront ディストリビューションを介してのみ S3 バケット内のコンテンツにアクセスできます。Word も参照してください。OACより詳細で強化されたアクセスコントロールを提供します。

ORR

「運用準備状況レビュー」を参照してください。

OT

「運用技術」を参照してください。

アウトバウンド (出力) 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 セキュリティリファレンスアーキテクチャでは、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VPCsを使用してネットワークアカウントを設定することをお勧めします。

P

アクセス許可の境界

ユーザーまたはロールが持つことができるアクセス許可の上限を設定するための Word プリンシパルにアタッチされる IAM 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、名前、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

「プログラム可能な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

アクセス許可の定義 (アイデンティティベースのポリシーを参照)、アクセス条件の指定 (リソースベースのポリシーを参照)、または の組織内のすべてのアカウントに対する最大アクセス許可の定義 AWS Organizations （ サービスコントロールポリシーを参照) が可能なオブジェクト。

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。詳細については、マイクロサービスでのデータ永続性の有効化 を参照してください。

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、「移行準備状況ガイド」を参照してください。

述語

`true` または を返すクエリ条件。通常は `false`WHERE句にあります。

述語プッシュダウン

転送前にクエリ内のデータをフィルタリングするデータベースクエリ最適化手法。これにより、リレーショナルデータベースから取得して処理する必要があるデータの量が減少し、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、Implementing security controls on AWSのPreventative controls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 のエンティティ。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「ロールの用語と概念」の「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じてプライバシーを考慮に入れたシステムエンジニアリングアプローチ。

プライベートホストゾーン

Amazon Route 53 が 1 つ以上の DNS 内のドメインとそのサブドメインの VPCs クエリに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

非準拠のリソースのデプロイを防止する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、プロビジョニング前にリソースをスキャンします。リソースがコントロールに準拠していない場合、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)

設計、開発、発売、成長と成熟、辞退と削除まで、ライフサイクル全体にわたる製品のデータとプロセスの管理。

本番環境

「環境」を参照してください。

プログラム可能なロジックコントローラー (PLC)

製造では、マシンをモニタリングし、製造プロセスを自動化する、信頼性が高く適応性の高いコンピュータです。

プロンプトの連鎖

1 つの LLM プロンプトの出力を次のプロンプトの入力として使用して、より良いレスポンスを生成します。この手法は、複雑なタスクをサブタスクに分割したり、事前対応を繰り返し調整または拡張したりするために使用されます。これにより、モデルのレスポンスの精度と関連性が向上し、より詳細でパーソナライズされた結果が得られます。

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。

publish/subscribe (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。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 では、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できるチャネルにイベントメッセージを発行できます。システムは、公開サービスを変更せずに新しいマイクロサービスを追加できます。

Q

クエリプラン

SQL リレーショナルデータベースシステム内のデータにアクセスするために使用する手順などの一連のステップ。

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

「責任者」、「説明責任者」、「相談先」、「通知先」 (RACI) を参照してください。

RAG

「取得拡張生成」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

「責任者」、「説明責任者」、「相談先」、「通知先」 (RACI) を参照してください。

RCAC

「行と列のアクセスコントロール」を参照してください。

リードレプリカ

読み取り専用に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。

再設計

「7 R」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)

サービスの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 R」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 とは分離され、独立しています。詳細については、AWS リージョン 「アカウントで使用できる を指定する」を参照してください。

回帰

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 R」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 R」を参照してください。

プラットフォーム変更

「7 R」を参照してください。

再購入

「7 R」を参照してください。

回復性

中断に耐えたり、中断から回復したりするアプリケーションの機能。で障害耐性を計画する場合、高可用性とディザスタリカバリがよく考慮されます AWS クラウド。詳細については、AWS クラウド「レジリエンス」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。

責任、説明責任、相談、情報提供 (RACI) 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。サポートを含めると、マトリックスは RASCI マトリックスと呼ばれ、除外すると RACI マトリックスと呼ばれます。

レスポンシブコントロール

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、Implementing security controls on AWSのResponsive controlsを参照してください。

保持

「7 R」を参照してください。

廃止

「7 R」を参照してください。

取得拡張生成 (RAG)

レスポンスを生成する前に、LLM が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を参照する生成 AI テクノロジー。例えば、RAG モデルは組織のナレッジベースやカスタムデータのセマンティック検索を実行する場合があります。詳細については、RAG とは」を参照してください。

ローテーション

攻撃者が認証情報にアクセスすることをより困難にするために、シークレットを定期的に更新するプロセス。

行と列のアクセスコントロール (RCAC)

アクセスルールが定義されている基本的で柔軟な SQL 式の使用。RCACは、行のアクセス許可と列マスクで構成されます。

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 (IdPs) が使用するオープンスタンダード。この機能により、フェデレーティ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AWS API で作成しなくても、 AWS Management Console にログインしたりIAM オペレーションを呼び出したりできます。SAML 2.0 ベースのフェデレーションの詳細については、Word 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

「監視コントロールとデータ収集」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 AWS Secrets Managerパスワードやユーザー認証情報などの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。シークレット値は、バイナリ、単一の文字列、または複数の文字列にすることができます。詳細については、Secrets Manager ドキュメントの「Secrets Manager シークレットの内容」を参照してください。

設計によるセキュリティ

開発プロセス全体を通じてセキュリティを考慮したシステムエンジニアリングアプローチ。

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、予防的、検出的、応答的、プロアクティブの 4 つの主なタイプがあります。

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。

セキュリティ情報とイベント管理 (SIEM) システム

セキュリティ情報管理 (SIM) システムとセキュリティイベント管理 (SEM) 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、その他のソースからデータを収集、監視、分析して、脅威やセキュリティ違反を検出し、アラートを生成します。

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に自動的に対応または修正するように設計された、事前定義され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の実装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。自動応答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、認証情報のローテーションなどがあります。

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。SCPsは、管理者がユーザーまたはロールに委任できるアクションのガードレールを定義するか、制限を設定します。SCPs を許可リストまたは拒否リストとして使用して、許可または禁止されるサービスまたはアクションを指定できます。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

のエントリポイントの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、スループットなど、サービスのパフォーマンス側面の測定。

サービスレベルの目標 (SLO)

サービスレベルのインジケータによって測定される、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、ユーザー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

セキュリティ情報とイベント管理システムを参照してください。

単一障害点 (SPOF)

システムを中断させる可能性のあるアプリケーションの 1 つの重要なコンポーネントの障害。

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。

split-and-seed 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、「」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 AWS クラウド」を参照してください。

SPOF

単一障害点を参照してください。

star スキーマ

トランザクションデータまたは測定データを保存するために 1 つの大きなファクトテーブルを使用し、データ属性を保存するために 1 つ以上の小さなディメンションテーブルを使用するデータベースの組織構造。この構造は、データウェアハウスまたはビジネスインテリジェンスの目的で使用するため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。このパターンを適用する方法の例については、「コンテナと Amazon ASP API Gateway を使用してレガシー Microsoft Word.NET (ASMX) ウェブサービスを段階的にモダナイズする」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ーゾーンに存在する必要があります。

監視コントロールとデータ収集 (SCADA)

製造では、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生産オペレーション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

ユーザーインタラクションをシミュレートして潜在的な問題を検出したり、パフォーマンスをモニタリングしたりする方法でシステムをテストします。Amazon CloudWatch Synthetics を使用してこれらのテストを作成できます。

システムプロンプト

動作を指示するために、コンテキスト、指示、またはガイドラインを LLM に提供するための手法。システムプロンプトは、コンテキストを設定し、ユーザーとのやり取りのルールを確立するのに役立ちます。

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の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

VPCs ネットワーク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トランジットハブ。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の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な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「 を他の 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 つのピザを食べることができる small 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 VPCs 間の接続。詳細については、Amazon VPC ドキュメントの「Word ピアリングとは」を参照してください。 VPC

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

現在のレコードに関連する行のグループに対して計算を実行する SQL 関数。ウィンドウ関数は、移動平均の計算や、現在の行の相対位置に基づく行の値へのアクセスなどの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

「書き込み 1 回」、「読み取り数」を参照してください。

WQF

AWS 「Word Workload Qualification Framework」を参照してください。

Write Once, Read Many (WORM)

データを 1 回書き込み、データの削除や変更を防ぐストレージモデル。許可されたユーザーは、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

ゼロデイエクスプロイト

ゼロデイ脆弱性を利用する攻撃、通常はマルウェア。

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

タスクを実行するための指示を LLM に提供しますが、タスクのガイドに役立つ例 (ショット) はありません。LLM は、事前にトレーニングされた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

CPU とメモリの平均使用量が 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。