ユーザーガイド

Amazon CodeCatalyst

Amazon CodeCatalyst: ユーザーガイド

Table of Contents

Amazon とは何ですか CodeCatalyst?

Amazon CodeCatalyst は、ソフトウェア開発プロセスに継続的インテグレーションとデプロイの方法を採用するソフトウェア開発チーム向けの統合サービスです。 CodeCatalyst 必要なツールがすべて 1 か所にまとめられています。継続的インテグレーション/継続的デリバリー (CI/CD) ツールを使用して、作業の計画、コードの共同作成、アプリケーションの構築、テスト、デプロイを行うことができます。スペースに接続することで、AWSリソースをプロジェクトと統合することもできます。AWS アカウント CodeCatalyst アプリケーションライフサイクルのすべての段階と側面を 1 つのツールで管理することで、ソフトウェアを迅速かつ自信を持って提供できます。

では CodeCatalyst、会社、部門、またはグループを代表するスペースを作成し、開発チームやタスクをサポートするのに必要なリソースを含むプロジェクトを作成します。CodeCatalystリソースは、スペース内にあるプロジェクト内で構成されます。チームがすぐに始められるように、CodeCatalyst 言語ベースまたはツールベースのプロジェクトブループリントを提供しています。プロジェクトブループリントからプロジェクトを作成すると、プロジェクトにはサンプルコードを含むソースリポジトリ、ビルドスクリプト、デプロイアクション、仮想サーバー、サーバーレスリソースなどのリソースが付属します。

何を使ってできるの? CodeCatalyst

CodeCatalyst 作業計画からアプリケーションのデプロイまで、ソフトウェア開発の各側面を、あなたと開発チームが実行できます。 CodeCatalystを使用してができます。

- コードの反復とコラボレーション — ソースコードリポジトリ内のブランチ、マージ、プルリクエスト、コメントを含むコードについて、チームと協力して作業できます。開発環境を作成して、リポジトリのクローンを作成したり、リポジトリへの接続を設定したりしなくても、コードをすばやく処理できます。
- ワークフローによるアプリケーションのビルド、テスト、デプロイ — ビルド、テスト、デプロイの各アクションを使用してワークフローを構成し、アプリケーションの継続的な統合と配信を処理します。ワークフローは手動で開始することも、コードプッシュやプルリクエストの作成または終了などのイベントに基づいて自動的に開始するように設定することもできます。
- 課題追跡でチームの作業に優先順位を付ける — 課題を使ってバックログを作成したり、進行中のタスクの状態をボードで監視したりできます。チームが取り組めるようなアイテムの健全なバックログを作成して維持することは、ソフトウェア開発の重要な部分です。
- 監視と通知の設定 — チームのアクティビティとリソースの状態を監視し、重要な変更が反映されるように通知を設定します。

どうやって始めればいいの CodeCatalyst？

スペースがない場合や、スペースの設定と管理の方法を知りたい場合は、Amazon CodeCatalyst 管理者ガイドから始めることをお勧めします。

プロジェクトやスペースでの作業に慣れていない場合は、以下から始めることをお勧めします。

- レビューする CodeCatalyst の概念
- スペースの作成
- の手順に従って最初のプロジェクトを作成する チュートリアル:モダン 3 層 Web アプリケーションブループリントによるプロジェクトの作成

詳細についてはこちらをご覧ください。 CodeCatalyst

の機能について詳しくは、 CodeCatalyst このユーザーガイドのほか、以下のリソースを参照してください。

- AWS DevOps Amazon に関するブログ記事 CodeCatalyst
- Amazon CodeCatalyst API リファレンスガイド
- Amazon CodeCatalyst アクション開発キット開発者ガイド
- CodeCatalyst よくある質問
- お客様の声

CodeCatalyst の概念

Amazon でのコラボレーションとアプリケーション開発を高速化するための主要な概念を理解してください CodeCatalyst。これらの概念には、出典管理、継続的インテグレーションと継続的デリバリー (CI/CD)、自動リリースプロセスのモデリングと設定に使用される用語が含まれます。

その他の概念情報については、以下のトピックを参照してください。

- ソースリポジトリの概念
- ワークフローの概念

トピック

- AWS のビルダー ID スペース CodeCatalyst
- で ID フェデレーションをサポートするスペース CodeCatalyst
- プロジェクト
- 設計図
- アカウント接続
- VPC 接続
- AWS ビルダー ID
- のユーザープロファイル CodeCatalyst
- ソースリポジトリ
- コミット
- 開発環境
- ワークフロー
- アクション
- 問題
- 個人用アクセストークン (PATs)
- 個人用接続
- ロール

AWS のビルダー ID スペース CodeCatalyst

スペース管理者は、メンバーページから個々の招待メールを送信 CodeCatalyst することで、ユーザーを に招待します。自分の AWS Builder ID CodeCatalyst を作成するために招待またはサインアップされたユーザー。プロファイルは AWS Builder ID で管理され、 のユーザー設定にユーザー名およびプロファイル情報として表示されます CodeCatalyst。

で ID フェデレーションをサポートするスペース CodeCatalyst

IAM Identity Center インスタンスの SSO ユーザーおよびグループに追加されたユーザーで、ID ストアで管理され、IAM Identity Center を介してスペースに招待されているユーザー。Space 管理者は、CodeCatalyst メンバーページを最新の更新と同期します。ユーザーは、会社の IAM Identity Center インスタンスで設定された SSO サインインポータルを使用してサインインします。ID フェデレーションをサポートするスペースは、Identity Center アプリケーションと ID ストア ID へのマッピングを介して ID ストアインスタンスに接続されます。

プロジェクト

プロジェクトは、開発チームやタスクをサポートする CodeCatalyst の共同作業を表します。プロジェクトを作成したら、ユーザーとリソースの追加、更新、削除、プロジェクトダッシュボードのカスタマイズ、チームの作業の進行状況のモニタリングを行うことができます。1 つのスペースに複数のプロジェクトを含めることができます。

プロジェクトの詳細については、「」を参照してくださいでプロジェクトを操作する CodeCatalyst。

設計図

ブループリントは、コンソールでプロジェクトを作成するとともに、アプリケーションサポートファイルと依存関係を生成して拡張する CodeCatalyst プロジェクトシンセサイザーです。で選択したブループリントからプロジェクトタイプを選択し CodeCatalyst、README ファイルを表示して、生成されるプロジェクトリポジトリとリソースをプレビューします。プロジェクトは、設計図で指定された基本設定から生成されます。プロジェクトのブループリントに定期的に合成します。これにより、ソフトウェアの依存関係などのプロジェクトファイルが更新され、リソースが再生成されます。プロジェクトは Projen というツールを使用して、最新のプロジェクト更新を同期し、サポートファイルを生成してプロジェクトを合成します。これらのファイルには`package.json`、アプリケーションタイプと言語に基づいて`Makefile`、`eslint`、、 などが含まれます。プロジェクトのブループリン

トでは、CDK コンストラクト、 AWS CloudFormation テンプレート、 AWS Serverless Application Model テンプレートなどの AWS リソースをサポートするファイルを生成できます。

プロジェクトのブループリントの詳細については、「」を参照してください CodeCatalyst ブループリントを使用した包括的なプロジェクトの作成。

アカウント接続

アカウント接続は、 CodeCatalyst スペースを に関連付けます AWS アカウント。アカウント接続がセットアップされると、 AWS アカウント がスペースで利用可能になります。その後、IAM ロールを に追加 CodeCatalyst して、 のリソースにアクセスできるようにします AWS アカウント。これらのロールを CodeCatalyst ワークフローアクションに使用することもできます。

アカウント接続の詳細については、「」を参照してください接続された AWS リソースへのアクセスを許可する AWS アカウント。

VPC 接続

VPC 接続は、ワークフローが VPC にアクセスするために必要なすべての設定を含む CodeCatalyst リソースです。スペース管理者は、スペースメンバーに代わって Amazon CodeCatalyst コンソールに独自の VPC 接続を追加できます。VPC 接続を追加することで、スペースメンバーはワークフローアクションを実行し、ネットワークルールに準拠し、関連付けられた VPC 内のリソースにアクセスできる開発環境を作成できます。

VPC 接続の詳細については、「 管理者ガイド」の「Amazon Virtual Private Cloud の管理」を参照してください。 CodeCatalyst

AWS ビルダー ID

AWS Builder ID は、 CodeCatalyst およびその他の参加アプリケーションへのサインアップとサインインに使用できる個人 ID です。これは とは異なります AWS アカウント。 AWS Builder ID は、ユーザーエイリアスや E メールアドレスなどのメタデータを管理します。Builder ID AWS は、 のすべてのスペースでユーザーをサポートする一意の ID です CodeCatalyst。 AWS Builder ID プロファイルへのアクセスについては、「」を参照してくださいプロフィールを更新する。 AWS Builder ID の詳細については、「」のAWS 「Builder ID」を参照してください AWS 全般のリファレンス。

サインアップとサインインの詳細については、「」を参照してくださいのセットアップと へのサインイン CodeCatalyst。

のユーザープロファイル CodeCatalyst

CodeCatalyst ユーザープロファイルにアクセスするには、 の任意のページでログインの初期化の下にあるドロップダウンからプロファイルオプションを選択します CodeCatalyst。プロファイルページから個人用アクセストークン (PATsを作成できますが、 を使用してのみ PATs を表示または削除できます AWS CLI。ユーザー名は、サインアップ時に選択したエイリアスです。ユーザー名を変更することはできません。別の CodeCatalyst ユーザーのプロファイルページを表示するには、プロジェクトのメンバータブに移動し、適切なユーザーを選択します。

Builder ID AWS にアクセスするには、 CodeCatalyst プロファイルを表示し、Builder ID AWS に移動します。Builder ID AWS プロファイルページにリダイレクトされます。プロファイルのフルネーム、E メールアドレス、パスワードは Builder ID AWS によって管理され、Builder ID AWS ページを使用してその情報を編集できます。サインアップ時にこの情報を入力しました。サインインに認証アプリケーションを使用するように MFA を設定する準備ができたら、 Builder ID AWS ページを使用します。 AWS Builder ID プロファイルの表示の詳細については、「」を参照してくださいプロフィールを更新する。

サインアップとサインインの詳細については、「」を参照してくださいのセットアップと へのサインイン CodeCatalyst。

ソースリポジトリ

ソースリポジトリは、プロジェクトのコードとファイルを安全に保存する場所です。また、ファイルのバージョン履歴も保存されます。デフォルトでは、ソースリポジトリは CodeCatalyst プロジェクトの他のユーザーと共有されます。1 つのプロジェクトに複数のソースリポジトリを持つことができます。でプロジェクトのソースリポジトリを作成することも CodeCatalyst、インストールされている拡張機能でそのサービスがサポートされている場合は、別のサービスによってホストされている既存のソースリポジトリをリンクすることもできます。例えば、 GitHub リポジトリ拡張機能をインストールした後、GitHub リポジトリをプロジェクトにリンクできます。詳細については、「のプロジェクトのリポジトリにソースコードを保存する CodeCatalyst」および「クイックスタート: 拡張機能のインストール、プロバイダーの接続、 でのリソースのリンク CodeCatalyst」を参照してください。

ソースリポジトリは、CI/CD ワークフローの属性とアクションを定義する設定ファイルなど、プロジェクトの設定 CodeCatalyst情報が保存される場所でもあります。ブループリントを使用してプロジェクトを作成すると、ソースリポジトリがプロジェクト設定情報とともに作成されます。空のプロジェクトを作成する場合は、ワークフローなどの設定情報を必要とするリソースを作成する前に、ソースリポジトリを作成する必要があります。

ソースリポジトリとソースコントロールの操作に役立つその他の概念については、「」を参照してくださいソースリポジトリの概念。

コミット

コミットとは、ファイルまたはファイルのセットに対する変更です。Amazon CodeCatalyst コンソールでは、コミットによって変更が保存され、ソースリポジトリにプッシュされます。コミットには、変更を行ったユーザーのアイデンティティ、変更日時、コミットタイトル、変更に関するメッセージなど、変更に関する情報が含まれます。詳細については、「Amazon でのコミットによるソースコードの変更について CodeCatalyst」を参照してください。

のソースリポジトリのコンテキストでは CodeCatalyst、コミットはリポジトリの内容に対する変更のスナップショットです。ユーザーが変更をコミットしてプッシュするたびに、 は、変更をコミットしたユーザー、コミットの日時、コミットの一部として行われた変更を含む情報 CodeCatalyst を保存します。コミットに Git タグを追加して、特定のコミットを識別することもできます。

コミットの詳細については、「」を参照してくださいAmazon でのコミットによるソースコードの変更について CodeCatalyst。

開発環境

開発環境は、 で使用 CodeCatalyst して、プロジェクトのソースリポジトリに保存されているコードをすばやく処理できるクラウドベースの開発環境です。開発環境に含まれるプロジェクトツールとアプリケーションライブラリは、プロジェクトのソースリポジトリ内の devfile によって定義されます。ソースリポジトリに devfile がない場合、デフォルトの devfile が自動的に適用されます。デフォルトの devfile には、最も頻繁に使用されるプログラミング言語とフレームワーク用のツールが含まれています。デフォルトでは、開発環境は 2 コアプロセッサ、4 GB の RAM、16 GiB の永続ストレージを持つように設定されています。

ワークフロー

ワークフローは、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) システムの一部としてコードを構築、テスト、デプロイする方法を説明する自動化された手順です。ワークフローは、ワークフローの実行中に実行する一連のステップまたはアクションを定義します。ワークフローは、ワークフローを開始するイベント、またはトリガー も定義します。ワークフローを設定するには、CodeCatalyst コンソールのビジュアルまたは YAML エディタ を使用してワークフロー定義ファイルを作成します。

> **Tip**
>
> プロジェクトでワークフローを使用する方法を簡単に確認するには、設計図 を使用してプロジェクトを作成します。各ブループリントは、レビュー、実行、および実験できる機能するワークフローをデプロイします。

ワークフローの詳細については、「でワークフローを使用して構築、テスト、デプロイする CodeCatalyst」を参照してください。

アクション

アクションはワークフローの主要な構成要素であり、ワークフローの実行中に実行する作業またはタスクの論理単位を定義します。通常、ワークフローには、設定方法に応じて順番または並行して実行される複数のアクションが含まれます。

アクションの詳細については、「」を参照してくださいワークフローが実行するアクションの設定。

問題

問題は、プロジェクトに関連する作業を追跡するレコードです。機能、タスク、バグ、またはプロジェクトに関連するその他の作業について問題を作成できます。アジャイル開発を使用している場合、問題はエピックやユーザーストーリーを記述することもできます。

問題の詳細については、「」を参照してくださいで問題のある作業を追跡して整理する CodeCatalyst。

個人用アクセストークン (PATs)

個人アクセストークン (PAT) はパスワードに似ています。これは、 のすべてのスペースとプロジェクトで使用できるようにユーザー ID に関連付けられています CodeCatalyst。PATs、統合開発環境 (IDEsと Git ベースのソースリポジトリを含む CodeCatalyst リソースにアクセスします。PATs でユーザーを表 CodeCatalyst し、ユーザー設定で管理できます。ユーザーは複数の PAT を持つことができます。個人用アクセストークンは 1 回だけ表示されます。ベストプラクティスとして、ローカルコンピュータに安全に保存してください。デフォルトでは、PATs 1 年後に期限切れになります。

PATs「」を参照してください個人用アクセストークンを使用してリポジトリアクセスをユーザーに付与する。

個人用接続

個人接続は、 CodeCatalyst アイデンティティと などの外部ソースプロバイダー間の認可です GitHub。個人用接続を使用して、 CodeCatalystユーザーがサードパーティーのソースリポジトリを追加できるようにします。例えば、 GitHub リポジトリを CodeCatalyst スペースに接続できます。インストールされているコネクタアプリケーションは、 GitHub アカウント所有者が指定したリポジトリで使用するため、アカウントにインストールされます。など、特定のプロバイダータイプのすべてのスペースで、1 つのユーザー ID (CodeCatalyst エイリアス) に対して 1 つの個人接続を作成できます GitHub。個人用接続は、ビルダー ID AWS または SSO ユーザーに関連付けられます。

詳細については、「個人接続による GitHub リソースへのアクセス」を参照してください。

ロール

ロールは、プロジェクトまたはスペースのリソースへのユーザーのアクセスと、ユーザーが実行できるアクションを定義します。ユーザーのロールは、プロジェクトに招待するときに選択します。にはスペースレベルのロールとプロジェクトレベルのロールがあります CodeCatalyst。適切なレベルの管理ロールを持つユーザーは、割り当てられたロールを変更できます。例えば、プロジェクトのプロジェクト管理者ロールを持つユーザーは、そのプロジェクトを完全に制御でき、そのプロジェクトのユーザーのロールを変更できます。使用可能なロールと各ロールに付与されているアクセス許可については、「」を参照してくださいユーザーロールによるアクセス許可の付与。

ロールの詳細については、「ユーザーロールによるアクセス許可の付与」をご参照ください。

のセットアップと へのサインイン CodeCatalyst

で設定できるスペースには、Builder ID ユーザーをサポートするスペースと ID AWS フェデレーションをサポートするスペースの 2 種類 CodeCatalystがあり、ここで SSO ユーザーとグループは IAM Identity Center で管理されます。Builder ID AWS スペースのユーザーは Builder ID CodeCatalyst で AWS にサインインし、エンタープライズスペースのユーザーはスペースに関連付けられた会社の SSO ポータル CodeCatalyst を使用して にサインインします。

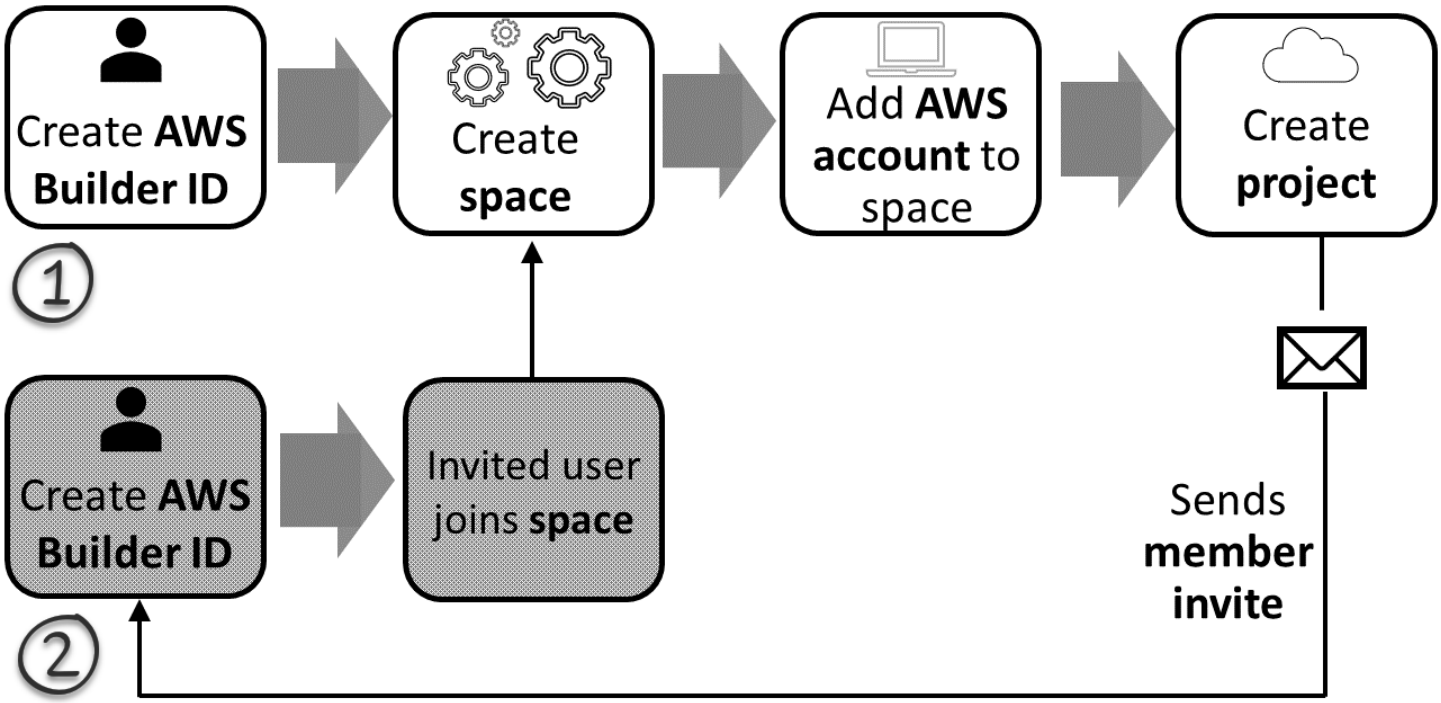
Builder ID AWS スペースをセットアップして管理する手順は、このガイドに記載されています。CodeCatalyst AWS Builder ID スペースを使用するには、 へのサインインに使用するユーザー設定と AWS Builder ID CodeCatalyst を使用して を設定します CodeCatalyst。

ID フェデレーションをサポートするスペースを設定および管理するためのステップは、「CodeCatalyst 管理者ガイド」に記載されています。ID フェデレーション用に設定されたスペースを操作するには、「Amazon CodeCatalyst 管理者ガイド」の CodeCatalyst 「スペースのセットアップと管理」を参照してください。

このセクションでは、Amazon で Builder ID スペース CodeCatalyst を使用するように を設定するための 2 AWS つの一般的なパスを提供します。1 人目のユーザーとしてスペースとプロジェクトを作成し、既存のスペースまたはプロジェクトへの招待を受け入れることです。これらのセットアップワークフローは、必ずしもかなり異なります。次の図は、両方のサインアッププロセスを次のように示しています。

Amazon で機能するように設定するための一般的なパスは 2 つあります CodeCatalyst。最初のユーザーとしてスペースとプロジェクトを作成し、既存のスペースまたはプロジェクトへの招待を受け入れることです。これらのセットアップワークフローは、必ずしもかなり異なります。次の図は、両方のサインアッププロセスを次のように示しています。

1. 最初のケースでは、会社、チーム、またはグループのスペースを作成して設定し、これらのリソースに他のユーザーを招待する前にプロジェクトを作成します。請求目的で を指定 AWS アカウント する必要があります。この場合も、 無料利用枠をデフォルトにすることができます。
2. 2 つ目のケースでは、プロジェクトへの招待を受け入れ CodeCatalyst て参加すると、別のユーザーが既にスペースとプロジェクトを作成しています。ただし、他のユーザーと連携する準備が整うようにプロファイルを設定する必要があります。



> **Tip**
>
> CodeCatalyst はスペースを使用してプロジェクトとリソースをグループ化します。に初めてサインアップすると CodeCatalyst、スペースとプロジェクトを作成するように求められます。

サインアップしてスペースとプロジェクトを作成する場合も、招待の承諾の一環としてサインアップする場合も、 への ログインに使用する AWS Builder ID を作成します CodeCatalyst。 AWS Builder ID を作成するには、 AWS アプリケーションへのサインインに使用するフルネーム、パスワード、E メールアドレスを指定します。この時点 CodeCatalyst 以降に にサインインするには、E メールとパスワードを使用します。この AWS Builder ID を使用して、Builder ID AWS 認証情報を使用する他のアプリケーションにログインすることもできます。

Builder ID の CodeCatalyst および AWS では、ログイン情報に基づいてプロファイルが生成されます。プロファイルには、 CodeCatalyst プロジェクト内の言語設定と通知設定 CodeCatalyst の設定が含まれています。

> **Tip**
>
> Amazon CodeCatalyst プロファイルへのサインアップ中に問題が発生した場合は、そのページに記載されている手順に従ってください。その他のヘルプが必要な場合は、「」を参照してくださいサインアップに関する問題。

トピック

最初のスペースと開発ロールの作成 (招待なしで開始)

既存のスペースやプロジェクトへの招待 CodeCatalyst なしで Amazon にサインアップできます。これを行うと、Builder ID AWS を作成した後にスペースとプロジェクトが作成されます。スペースの作成の一環として、請求 AWS アカウント 目的で を追加する必要があります。

> **Tip**
>
> Amazon CodeCatalyst プロファイルへのサインアップ中に問題が発生した場合は、そのページに記載されている手順に従ってください。その他のヘルプが必要な場合は、「」を参照してくださいサインアップに関する問題。

プロジェクトやスペースへの招待 CodeCatalyst なしで から開始するユーザーに考えられるフローの 1 つを次に示します。

Mary Major は、関心があり CodeCatalyst 、試してみることに決める開発者です。 CodeCatalyst コンソールに移動し、サインアップして Builder ID AWS を作成するオプションを選択します。Mary は、Builder ID AWS を作成するための E メールアドレスとパスワードを提供します。ビルダー ID

AWS を使用して CodeCatalyst や他のアプリケーションにサインインできるようになります。エイリアスを選択するように求められたら、 に表示される CodeCatalyst ユーザー名`MaryMajor`としてを指定 CodeCatalyst し、他のプロジェクトメンバーが @mention Mary に使用するように指定します。

次に、Mary は自動的にスペースを作成するように指示されます。このフローの一環として、Mary は、最初のプロジェクトの構築とデプロイでサンプルコードを表示できるように、作成している AWS アカウント スペースに を関連付けるよう求められます。その情報を追加してスペースを作成します。そこで、新しいスペースのプロジェクトに使用できるプレビュー開発ロールを作成するオプションを選択します。Mary はプロジェクトの作成を選択し、プロジェクトのブループリントのリストを表示します。利用可能なブループリントの情報を確認した後、最初のプロジェクトで Modern 3 層ウェブアプリケーションブループリントを試すことにしました。必須フィールドに入力し、プロジェクトを作成します。プロジェクトの準備ができるとすぐに、最近のアクティビティと、そのコードを自動的に構築してデプロイするプロジェクトコードとワークフローへのリンクを含むプロジェクト概要ページが表示されます。デプロイされたサンプルウェブアプリケーションの表示を含め、コードとワークフローの両方を調べます。見ているものを気にして、同僚の一部をプロジェクトに招待し、 の探索を開始することにしました CodeCatalyst。

しばらくすると、Mary は多要素認証 (MFA) CodeCatalyst で にサインインするように AWS Builder ID を設定します。MFA を設定する CodeCatalyst と、Mary は自分の CodeCatalyst パスワードと、承認されたサードパーティー認証アプリケーションからのパスコードまたはトークンを組み合わせてにサインインできます。

最初のスペースと IAM ロールの作成

Amazon CodeCatalyst プロファイルにサインアップし、スペースを作成し、スペースのアカウント、サポートロール、およびデベロッパーロールを追加するには、次の手順に従います。

最後の手順では、開発者ロールを作成して追加します。デベロッパーロールは、 AWS CodeCatalyst ワークフローが リソースにアクセス AWS できるようにする IAM ロールです。デベロッパーロールは、 の管理に使用されるサービスロールであり、サインインしたアカウントで AWS のサービス 作成されます。サービスロールとは、サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除できます。ロールには という名前が付けられますCodeCatalystWorkflowDevelopmentRole-*spaceName*。ロールとロールポリシーの詳細については、「」を参照してくださいCodeCatalystWorkflowDevelopmentRole-***spaceName*** サービスロールについて。

> **Note**
>
> セキュリティのベストプラクティスとして、 スペース内の AWS リソースへのアクセスを管理する必要がある管理ユーザーと開発者にのみ管理アクセスを割り当てます。

開始する前に、管理者権限を持つアカウントの AWS アカウント ID を提供する準備が整っている必要があります。12 桁の AWS アカウント ID を用意します。 AWS アカウント ID の検索については、AWS アカウント 「ID とそのエイリアス」を参照してください。

新規ユーザーとしてサインアップするには

1. CodeCatalyst コンソールで開始する前に、 を開き AWS Management Console、スペースの作成 AWS アカウント に使用するのと同じ でサインインしていることを確認します。
2. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
3. [Welcome] ページで、[サインアップ] を選択します。[AWS ビルダー ID の作成] ページが表示されます。Builder ID AWS は、サインインするために作成する ID です。これは とは異なります AWS アカウント。
4. E メールアドレス に、 に関連付ける E メールアドレスを入力します CodeCatalyst。次いで、[次へ] を選択します。
5. 名前 で、 AWS ビルダー ID を使用するアプリケーションに表示する姓名を指定します。スペースは許可されます。これは、Mary Major などの AWS Builder ID プロファイル名になります。名前は後で変更できます。

   [次へ] をクリックします。E メール検証ページが表示されます。
6. 指定した E メールに検証コードが送信されます。このコードを検証コード に入力し、検証 を選択します。5 分経ってもコードが届かず、スパムフォルダや迷惑メールフォルダで見つからない場合は、コードの再送信 を選択します。
7. コードを確認したら、パスワード とパスワードの確認 の要件を満たすパスワードを入力します。

   AWS カスタマーアグリーメントおよび AWS サービス条件への同意を確認するチェックボックスをオンにし、 AWS ビルダー ID の作成 を選択します。
8. エイ CodeCatalyst リアスの作成ページで、一意のユーザー識別子に使用するエイリアスを に入力します CodeCatalyst。など、スペースなしで名前の短縮バージョンを選択しますMaryMajor。 CodeCatalyst 他のユーザーはこれを使用して、コメントやプルリクエストで

ユーザーを @mention します。 CodeCatalyst プロファイルには、ビルダー ID AWS のフルネームと CodeCatalyst エイリアスの両方が含まれます。エイ CodeCatalyst リアスは後で変更できません。

フルネームとエイリアスは、 のさまざまなエリアに表示されます CodeCatalyst。例えば、アクティビティフィードにリストされているアクティビティのプロファイル名が表示されますが、プロジェクトメンバーはエイリアスを使用して @mention します。

[次へ] をクリックします。ページが更新され、 CodeCatalyst 「スペースの作成」セクションが表示されます。

9. 「スペースの名前」に、スペースの名前を入力します。これは後で変更することはできません。

> **Note**
>
> スペース名は 全体で一意である必要があります CodeCatalyst。削除されたスペースの名前は再利用できません。

10. AWS リージョン ドロップダウンメニューで、スペースとプロジェクトデータを保存するリージョンを選択します。これは後で変更することはできません。
11. [次へ] をクリックします。ページが更新され、 を追加するページが表示されます AWS アカウント。このアカウントは、スペースの請求アカウントとして使用されます。
12. AWS アカウント ID に、スペースに接続するアカウントの 12 桁の ID を入力します。

    [AWS アカウント確認トークン] に、生成されたトークン ID をコピーします。トークンは自動的にコピーされますが、 AWS 接続リクエストの承認中に保存することもできます。
13. AWS コンソールに移動 を選択して を確認します。
14. で「Amazon CodeCatalyst スペースの検証」ページが開きます AWS Management Console。これは Amazon CodeCatalyst スペースページです。ページにアクセスするには、サインインが必要な場合があります。

    で AWS Management Console、スペースを作成する AWS リージョン 場所と同じ を選択してください。

    ページに直接アクセスするには、https://console.aws.amazon.com/codecatalyst/home/. の で Amazon CodeCatalyst Spaces AWS Management Console にサインインします。

    の検証トークンフィールド AWS Management Console には、 で生成されたトークンが自動的に入力されます CodeCatalyst。

15. （オプション）「承認された有料利用枠」で「有料利用枠の承認 (スタンダード、エンタープライズ) 」を選択して、請求アカウントの有料利用枠を有効にします。

> **Note**
>
> これにより、請求階層が有料階層にアップグレードされることはありません。ただし、では、スペースの請求階層をいつでも変更 AWS アカウント できるように が設定されます CodeCatalyst。有料利用枠はいつでも有効にできます。この変更を行わない場合、スペースは 無料利用枠のみを使用できます。

16. [スペースを確認] を選択します。

    アカウントがスペースに追加されたことを示すアカウント検証の成功メッセージが表示されます。

17. Amazon CodeCatalyst スペースの検証ページにとどまります。次のリンクを選択します。このスペースに IAM ロールを追加するには、スペースの詳細を表示します。

    CodeCatalyst でスペースの詳細を含む接続ページが開きます AWS Management Console。これは Amazon CodeCatalyst スペースページです。ページにアクセスするには、ログインが必要な場合があります。

18. CodeCatalyst ページに戻り、次へ を選択します。

19. スペースの作成中は、ステータスメッセージが表示されます。スペースが作成されると、CodeCatalyst 次のメッセージが表示されます。スペースの準備ができました。最後のステップはプロジェクトの作成です。次のいずれかを試すことができます。

    - 今すぐスキップ を選択します。
    - スペースの最初のプロジェクトを作成するを選択します。設計図を使用してプロジェクトを作成する方法を示すチュートリアルについては、「」を参照してください。 チュートリアル:モダン 3 層 Web アプリケーションブループリントによるプロジェクトの作成

> **Note**
>
> アクセス許可エラーまたはバナーが表示された場合は、ページを更新してページをもう一度表示してみてください。

を作成して追加するには CodeCatalyst CodeCatalystWorkflowDevelopmentRole-*spaceName*

1. CodeCatalyst コンソールで開始する前に、 を開き AWS Management Console、 AWS アカウント スペースに対して同じ でログインしていることを確認します。
2. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
3. CodeCatalyst スペースに移動します。[Settings (設定)]、[AWS アカウント] の順に選択します。
4. ロール AWS アカウント を作成する のリンクを選択します。AWS アカウント 詳細ページが表示されます。
5. からロールの管理を選択します AWS Management Console。

   で「Amazon CodeCatalyst スペースへの IAM ロールの追加」ページが開きます AWS Management Console。これは Amazon CodeCatalyst スペーススペースです。ページにアクセスするには、ログインが必要な場合があります。
6. IAM で CodeCatalyst 開発管理者ロールの作成を選択します。このオプションは、開発ロールのアクセス許可ポリシーと信頼ポリシーを含むサービスロールを作成します。ロールには という名前が付けられますCodeCatalystWorkflowDevelopmentRole-*spaceName*。ロールとロールポリシーの詳細については、「」を参照してくださいCodeCatalystWorkflowDevelopmentRole-*spaceName* サービスロールについて。

   > **Note**
   >
   > このロールはデベロッパーアカウントでのみ使用が推奨され、AdministratorAccess AWS 管理ポリシーを使用して、この で新しいポリシーとリソースを作成するためのフルアクセスを付与します AWS アカウント。
7. 開発ロールの作成 を選択します。
8. 接続ページの で使用できる IAM ロールで CodeCatalyst、アカウントに追加された IAM CodeCatalystWorkflowDevelopmentRole-*spaceName*ロールのリストでロールを表示します。
9. スペースに戻るには、「Amazon に移動 CodeCatalyst」を選択します。

を作成して追加するには CodeCatalyst AWSRoleForCodeCatalystSupport

1. CodeCatalyst コンソールで開始する前に、 を開き AWS Management Console、 AWS アカウント スペースに対して同じ でログインしていることを確認します。
2. CodeCatalyst スペースに移動します。[Settings (設定)]、[AWS アカウント] の順に選択します。

3. ロール AWS アカウント を作成する のリンクを選択します。AWS アカウント 詳細ページが表示されます。
4. からロールの管理を選択します AWS Management Console。

   で「Amazon CodeCatalyst スペースへの IAM ロールの追加」ページが開きます AWS Management Console。これは Amazon CodeCatalyst Spaces ページです。ページにアクセスするには、サインインが必要な場合があります。
5. CodeCatalyst スペースの詳細 で、 CodeCatalyst サポートロールの追加 を選択します。このオプションは、プレビュー開発ロールのアクセス許可ポリシーと信頼ポリシーを含むサービスロールを作成します。ロールには一意の識別子AWSRoleForCodeCatalystSupportが付加された名前が付けられます。ロールとロールポリシーの詳細については、「」を参照してくださいAWSRoleForCodeCatalystSupport サービスロールについて。
6. CodeCatalyst サポート用のロールを追加 ページで、デフォルトを選択したまま、ロールの作成を選択します。
7. で使用できる IAM ロール CodeCatalystで、アカウントに追加された IAM `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`*ロールのリストでロールを表示します。
8. スペースに戻るには、「Amazon に移動 CodeCatalyst」を選択します。

Builder ID AWS を作成して最初のスペースを作成し、アカウントを追加したら、プロジェクトを作成できます。詳細については、「「プロジェクトの作成」」を参照してください。を初めて使用する場合は CodeCatalyst、 から始めることをお勧めしますチュートリアル:モダン 3 層 Web アプリケーションブループリントによるプロジェクトの作成。

招待を受け入れて Builder ID AWS を作成する

プロジェクトまたはスペースへの招待を受け入れる一環 CodeCatalyst として、Amazon にサインアップできます。招待を承諾する一環として、ビルダー ID AWS を作成するように求められます。Builder ID AWS を使用して、 のリソースにアクセスします CodeCatalyst。

> **Tip**
>
> その他のヘルプが必要な場合は、「」を参照してくださいサインアップに関する問題。

プロジェクトまたはスペースへの招待 CodeCatalyst で から始めるユーザーに対して考えられるフローの 1 つを次に示します。

Saanvi Sarkar は、 CodeCatalyst プロジェクト管理者としてプロジェクトに参加する招待を受け取ったデベロッパーです。Saanvi は招待を受け入れ、 のサインインページを開きます CodeCatalyst。サインアップすることを選択し、Builder ID を作成するための E AWS メールアドレスとパスワードを提供します。Saanvi は、自分の AWS Builder ID を使用して CodeCatalyst や他のアプリケーションにサインインできるようになります。後で、自分のプロファイルを編集して、ログイン用 E メールアドレスまたはパスワードを変更できます。エイリアスを選択するように求められた場合、Saanvi は に表示される CodeCatalyst エイリアス`SaanviSarkar`として を指定 CodeCatalyst し、他のプロジェクトメンバーが @mention Saanvi に使用するように指定します。サインアップすると、Saanvi は AWS Builder ID 認証情報を使用する他のアプリケーションにサインイン認証情報を使用できるようになります。

サインアップが完了すると、Saanvi は招待で指定された CodeCatalyst プロジェクトとスペースに自動的に参加します。また、この招待は、プロジェクトとスペースでのロールに対する事前定義されたアクセス許可も提供します。プロジェクト設定では、Saanvi のエイリアスは、割り当てられたプロジェクトロールを持つメンバーリストに表示されます。でソースリポジトリを操作するために CodeCatalyst、Saanvi は少し時間をとって個人アクセストークン (PAT) を作成します。PAT は、ソースの変更や認証トークンを必要とするアクションを行うときに、 で認証 CodeCatalyst に使用されます。

Saanvi がプロジェクトで作業する場合、そのエイリアスはプロジェクトの作業アクティビティログに一覧表示されます。Saanvi の問題とコメントには、他のプロジェクトメンバーが返信で @mention できるエイリアスが表示されます。別のプロジェクトメンバーを @mention するために、Saanvi は CodeCatalyst プロファイルでエイリアスを検索します。

しばらくすると、Saanvi は多要素認証 (MFA) CodeCatalyst で にサインインするように AWS Builder ID を設定します。MFA を設定する CodeCatalyst と、Saanvi は CodeCatalyst パスワードと、承認されたサードパーティー認証アプリケーションからのパスコードまたはトークンを組み合わせて にサインインできます。

招待の承諾と Builder ID AWS の作成

Amazon のプロジェクトまたはスペースに招待されると CodeCatalyst、notify@codecatalyst.aws から招待を受け入れるよう求めるメールが届きます。 AWS ビルダー ID が既にあり、 にサインインしている場合 CodeCatalyst、招待を受け入れるを選択すると、ブラウザタブでプロジェクトまたはスペースが自動的に開きます。コンソールにサインインしていないが Builder ID AWS がある場合は、

サインインページが表示されます。詳細については、「AWSビルダー ID でサインイン」を参照してください。

AWS Builder ID がない場合は、招待を受け入れるを選択するとサインインページが表示され、Builder ID AWS を作成するオプションを選択する必要があります。

招待を受け入れて Builder ID AWS を作成するには

1. 招待 E メールで、招待を受け入れる を選択します。
2. サインインページで、「サインアップしない」を選択します。 AWS Builder ID を作成します。

   > **Tip**
   >
   > Builder ID AWS は、サインインするために作成する ID です。これは とは異なります AWS アカウント。

3. AWS 「ビルダー ID の作成」ページの「E メールアドレス」に、ビルダー ID に使用する E AWS メールアドレスを入力します。

   名前 で、 AWS ビルダー ID を使用するアプリケーションに表示する姓名を指定します。スペースは許可されます。これは、Mary Major などの AWS Builder ID プロファイル名になります。名前は後で変更できます。

   [次へ] をクリックします。

   指定した E メールに検証コードが送信されます。このコードを検証コード に入力し、検証 を選択します。5 分経ってもコードが届かず、スパムフォルダまたは迷惑メールフォルダで見つからない場合は、コードの再送信 を選択します。
4. コードが検証されたら、パスワード とパスワードの確認 の要件を満たすパスワードを入力します。
5. AWS ビルダー ID の作成 を選択します。
6. エイリアスの作成ページで、一意のユーザー識別子に使用するエイリアスを に入力します CodeCatalyst。など、スペースなしで名前の短縮バージョンを選択しますMaryMajor。CodeCatalyst 他のユーザーはこれを使用して、コメントやプルリクエストでユーザーを @mention します。 CodeCatalyst プロファイルには、ビルダー ID AWS のフルネームと CodeCatalyst エイリアスの両方が含まれます。 CodeCatalyst エイリアスは変更できません。

フルネームとエイリアスは、 のさまざまなエリアに表示されます CodeCatalyst。例えば、アクティビティフィードにリストされたアクティビティのプロファイル名が表示されますが、プロジェクトメンバーはエイリアスを使用して @mention します。

[エイリアスの作成] を選択します。招待されたプロジェクトまたはスペースに移動します。

AWSビルダー ID でサインイン

Amazon CodeCatalyst プロフィールにサインインするには、次の手順に従います。

> **Note**
>
> 多要素認証 (MFA) 用のデバイスをもう登録しましたか? セキュリティを強化するために、Amazon CodeCatalyst で MFA を設定することを強くお勧めします。詳細については、「多要素認証用のデバイスを登録する方法」を参照してください。

AWS Builder ID でサインインするには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. E メールアドレスを入力します。今後のサインインに備えてメールアドレスを保存したい場合は、[メールアドレスを保存] をオプションで選択します。[続行] を選択します。
3. [Password] (パスワード) を入力します。[Sign in (サインイン)] を選択します。パスワードを思い出せない場合は、の手順に従ってください。パスワードを忘れてしまいました

信頼されたデバイス

サインインページで「これは信頼できるデバイスです」オプションを選択すると、Amazonはそのデバイスからのfuture CodeCatalyst サインインをすべて承認済みと見なします。信頼できるデバイスを使用している限り、Amazon CodeCatalyst は MFA コードを入力するオプションを提示しません。例外として、新しいブラウザからのサインインや、デバイスに未知の IP アドレスが発行された場合などがあります。

SSO でサインイン

SSO を使用して Amazon CodeCatalyst にサインインするには、次の手順に従います。

代わりに AWS Builder ID を使用してサインインする方法については、を参照してくださいAWSビルダー ID でサインイン。

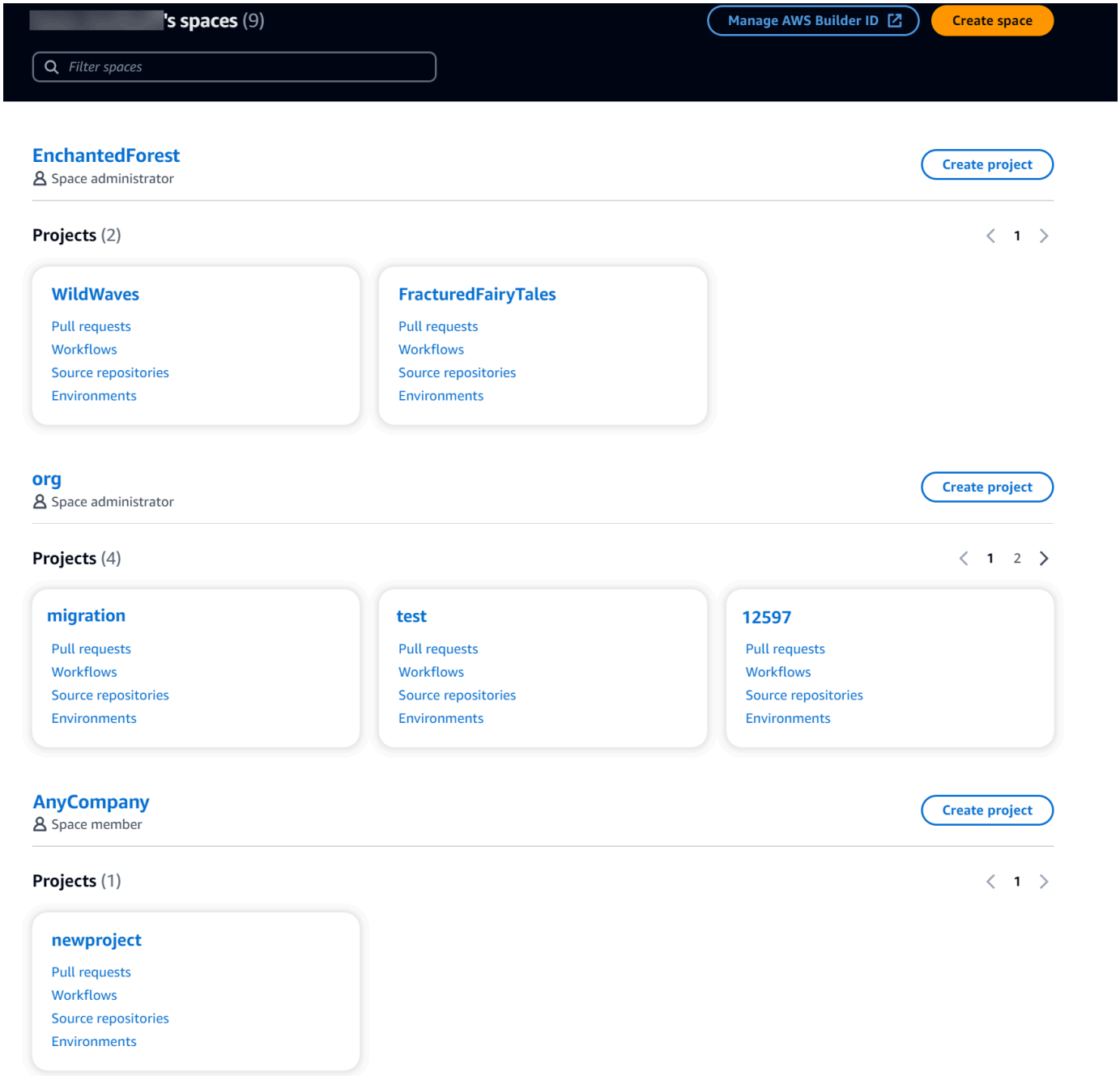
SSO でサインインするには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. [サインインオプションの選択] で [シングルサインオン (SSO) を使用] を選択します。
3. [AWSIdentity Center アプリケーション名] に、ID フェデレーション管理者から提供されたアプリケーション名を入力します。
4. 「IAM ID センターに進む」を選択します。

ユーザーのすべてのスペースとプロジェクトを表示する

スペースとプロジェクトのリストは、ユーザーのホームページに表示できます。ユーザーホームページには、ユーザーが所属する各スペースのリスト、そのスペース内のユーザーの役割 (スペース管理者など)、およびユーザーがメンバーシップを持っている各スペースのプロジェクトが表示されます。

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. ブラウザに次のアドレスを入力します。https://codecatalyst.aws/home



3. 開きたいスペースまたはプロジェクトを選択します。期待していたスペースやプロジェクトが表示されない場合は、別のユーザーとしてサインインする必要がある場合があります。

CodeCatalyst プロフィールの表示と管理

Amazon CodeCatalyst でユーザープロファイルを表示して、E CodeCatalyst メールアドレスやエイリアスなどの情報を取得できます。プロファイルと AWS Builder ID を更新することもできます。パスワードを忘れた場合は、パスワードのリセットをリクエストできます。

CodeCatalyst プロフィールを表示する

サインアップ時に提供した情報は、Amazon CodeCatalyst にログインするための認証情報として使用され、プロファイルで管理されます。これには、名前、ニックネーム、サインインに使用するメールアドレスが含まれます。 CodeCatalyst

> **Note**
>
> AWSBuilder ID のニックネームはエイリアスではありません。 CodeCatalyst CodeCatalyst サインアップ時にエイリアスを選択しました。

CodeCatalyst プロフィールを表示するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. 右上の、最初のイニシャルが付いたアイコンの横にある矢印を選択し、[マイセッティング] を選択します。[CodeCatalyst マイセッティング] ページが開きます。
3. AWSBuilder ID のメールアドレスまたはパスワードを更新したり、MFA を設定したりするには、「AWSBuilder ID を管理」を選択します。AWSビルダー ID ページが開きます。

CodeCatalyst別のユーザーのプロフィールを表示する

CodeCatalyst 他のユーザーのプロフィールを表示するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. サイド・ナビゲーションで「プロジェクト設定」を選択します。「メンバー」タブを選択します。 CodeCatalystプロジェクトのメンバーのリストを表示します。
3. 調べたいメンバー名または @mention を選択してください。マイセッティングページには、ユーザーのエイリアス、メールアドレス、フルネームが表示されます。@mention CodeCatalyst プロジェクトメンバーにはエイリアスを使用してください。

> **Note**
>
> AWSユーザーのビルダー ID CodeCatalystニックネームはエイリアスではありません。CodeCatalyst 登録時にエイリアスを選択しました。

プロジェクト内の他のユーザーのプロフィールを表示するには、リストでそのユーザーの名前を選択します。

プロフィールを更新する

では CodeCatalyst、プロフィールは AWSBuilder ID によって管理される個人情報と、で管理される設定で構成されています CodeCatalyst。

- プロフィールのフルネーム、メールアドレス、パスワードは AWSBuilder ID によって管理されます。この情報はサインアップ時に入力しました。アプリケーションのサインインに認証アプリを使用するように MFA を設定すると、 CodeCatalyst AWSビルダー ID ページが表示されます。
- CodeCatalyst 個人アクセストークン (PAT)、 CodeCatalyst 通知、言語設定の設定は、の「My settings」ページで管理されます。 CodeCatalyst詳細については、「個人用アクセストークンを使用してリポジトリアクセスをユーザーに付与する」を参照してください。

> **Note**
>
> AWSBuilder ID のフルネーム (CodeCatalyst ディスプレイネーム) とファーストネームを更新できます。ただし、 CodeCatalyst エイリアスは変更できません。

AWSビルダー ID またはメールアドレスの更新

AWS ビルダー ID自分またはメールアドレスを更新するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. 右上の、最初のイニシャルが付いたアイコンの横にある矢印を選択し、[マイセッティング] を選択します。[CodeCatalystマイセッティング] ページが開きます。
3. プロフィールページで「AWSビルダー ID を管理」を選択します。「AWSビルダー ID」ページが開きます。
4. ページの左側で [My details] を選択します。
5. [プロフィール情報] で [編集] を選択し、名前またはニックネームを更新します。ニックネームを指定しなかった場合、ニックネームフィールドにはフルネームのファーストネームが反映されます。これはあなたのエイリアスではありません。 CodeCatalyst

> **Note**
>
> これにより AWS Builder ID のフルネームとファーストネームが更新されます。CodeCatalyst エイリアスは更新されません。

[連絡先情報] で [編集] を選択し、メールアドレスを更新します。

> **Note**
>
> これにより、サインインに使用するメールアドレスが更新されます CodeCatalyst。

CodeCatalyst パスワードの変更

CodeCatalyst パスワードを変更するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. 右上の、最初のイニシャルが付いたアイコンの横にある矢印を選択し、[ユーザープロフィール] を選択します。[CodeCatalyst マイセッティング] ページが開きます。
3. プロフィールページで「AWSビルダー ID を管理」を選択します。「AWSビルダー ID」ページが開きます。
4. ページの左側で [セキュリティ] を選択します。
5. [パスワードを変更] を選択し、指示に従います。

AWS CLIとを使用するためのセットアップ CodeCatalyst

Amazon CodeCatalyst コンソールは、日常のほとんどのタスクを処理する場所です。ただし、Dev Environments、個人アクセストークン、AWS CLI CodeCatalystまたはイベントのログインを操作する場合は、の設定と設定が必要になることがあります。で使用する前に、AWS CLIプロファイルをインストールして設定する必要があります。 CodeCatalyst

AWS CLIをセットアップするには CodeCatalyst

1. AWS CLI の最新バージョンをインストールします。AWS CLIのバージョンが既にインストールされている場合は、そのバージョンが最新であり、のコマンドが含まれていることを確認し

CodeCatalyst、必要に応じて更新してください。 CodeCatalyst コマンドを含むバージョンがインストールされていることを確認するには、コマンドプロンプトを開いて以下のコマンドを実行します。

```
aws codecatalyst help
```

CodeCatalyst コマンドのリストが表示されたら、 CodeCatalystそのバージョンがサポートされています。コマンドが認識されない場合は、AWS CLIのバージョンを最新バージョンに更新してください。詳細については、『AWS Command Line Interfaceユーザガイド』の「の最新バージョンをインストールまたは更新する」を参照してください。AWS CLI

2. プロファイルがない場合や、専用の名前付きプロファイルを使用する場合は、aws configureコマンドを実行してプロファイルを作成します CodeCatalyst。専用の名前付きプロファイルを作成することをお勧めしますが CodeCatalyst、デフォルトプロファイルを使用することもできます。詳細については、「設定の基本」を参照してください。
3. configプロファイルのファイルを編集し、 CodeCatalyst 接続用のセクションを次のように追加します。「config」ファイルは、Linux または macOS では「~/.aws/config」、Windows では「C:\Users*USERNAME*\.aws\config」にあります。

```
[profile codecatalyst]
region = us-west-2
sso_session = codecatalyst

[sso-session codecatalyst]
sso_region = us-east-1
sso_start_url = https://view.awsapps.com/start
sso_registration_scopes = codecatalyst:read_write
```

4. ファイルを保存します。
5. コマンドを実行する前に、 CodeCatalyst 新しいターミナルまたはコマンドプロンプトを開き、次のコマンドを実行して、aws codecatalystコマンドを実行するための認証情報を要求および取得します。codecatalyst必要に応じてプロファイルの名前に置き換えてください。

```
aws sso login --profile codecatalyst
```

codecatalystコマンドの例を表示するには、以下のトピックを参照してください。

- 個人用アクセストークンを使用してリポジトリアクセスをユーザーに付与する

- イベントログを使用したログイベントへのアクセス

入門チュートリアル

Amazon CodeCatalyst では、プロジェクトの開始に役立つさまざまなテンプレートを用意しています。空のプロジェクトから始めて、そのプロジェクトにリソースを追加することもできます。これらのチュートリアルの手順に従って、作業方法をいくつか学んでください。 CodeCatalyst

初めて使用する場合は CodeCatalyst、チュートリアル:モダン 3 層 Web アプリケーションブループリントによるプロジェクトの作成から始めることをお勧めします。

> **Note**
>
> これらのチュートリアルに従うには、まずセットアップを完了する必要があります。詳細については、「のセットアップと へのサインイン CodeCatalyst」を参照してください。

トピック

- チュートリアル:モダン 3 層 Web アプリケーションブループリントによるプロジェクトの作成
- チュートリアル:空のプロジェクトから始めて、手動でリソースを追加する
- チュートリアル: CodeCatalyst 生成 AI 機能を使用して開発作業を高速化する
- チュートリアル: 構成可能な PDK ブループリントを使用したフルスタックアプリケーションの作成

の特定の機能分野に焦点を当てたその他のチュートリアルについては CodeCatalyst、以下を参照してください。

- Slack 通知を使い始める
- CodeCatalyst ソースリポジトリと単一ページのアプリケーション設計図の開始方法
- ワークフローの開始方法
- カスタムブループリントの開始方法
- Amazon CodeCatalyst アクションデベロッパーガイドを使って始めましょう

詳細なチュートリアルについては、以下を参照してください。

- チュートリアル: Amazon S3 にアーティファクトをアップロードする

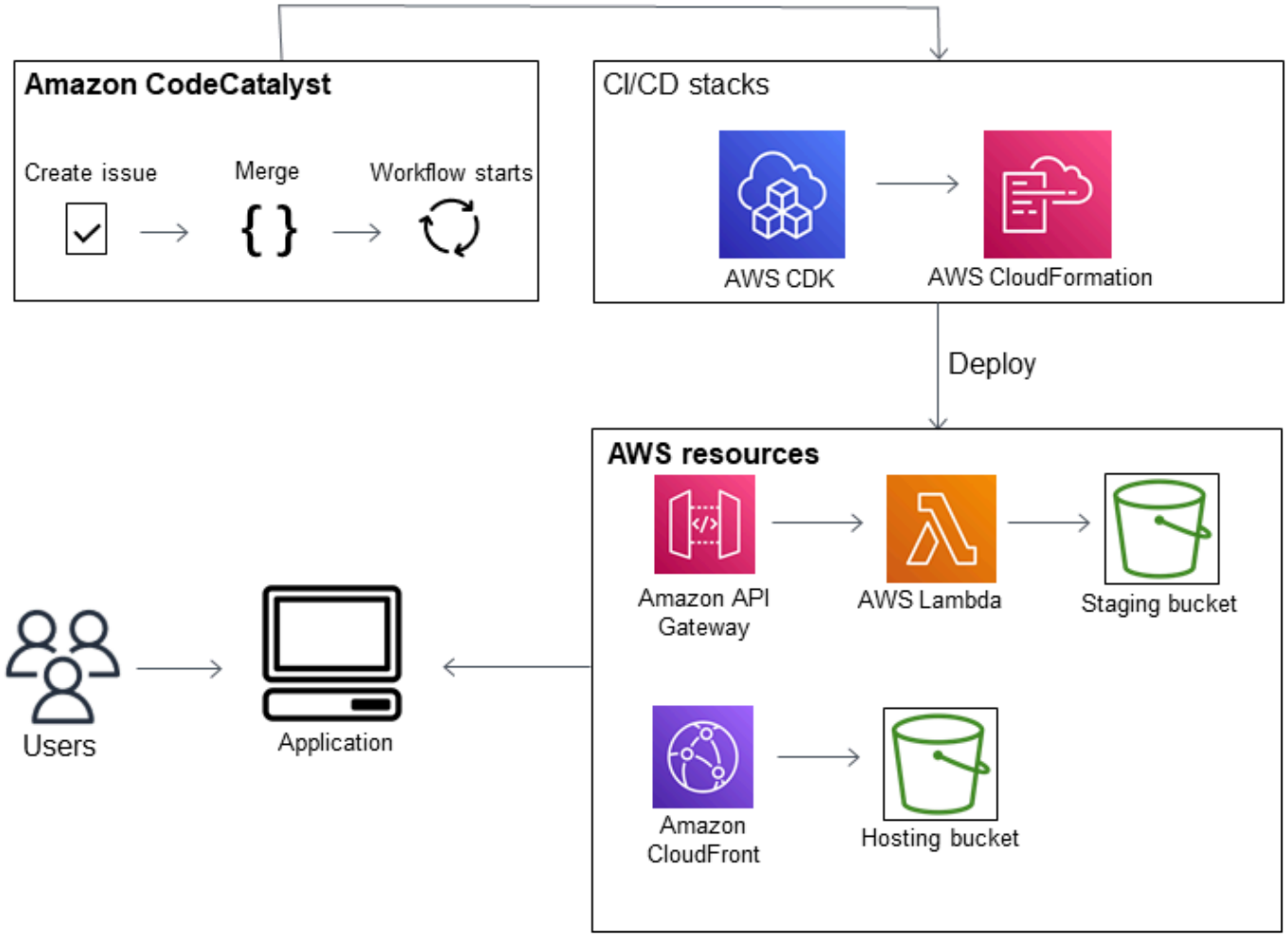
- チュートリアル: を使用してサーバーレスアプリケーションをデプロイする AWS CloudFormation
- チュートリアル: Amazon ECS にアプリケーションをデプロイする
- チュートリアル: Amazon EKS にアプリケーションをデプロイする
- チュートリアル: ワークフローでアクションを使用して GitHubコードをリントする
- チュートリアル:React アプリケーションの作成と更新

チュートリアル:モダン 3 層 Web アプリケーションブループリントによるプロジェクトの作成

ブループリントを使用してプロジェクトを作成すると、ソフトウェア開発をより早く開始できます。ブループリントを使用して作成されたプロジェクトには、コードを管理するためのソースリポジトリや、アプリケーションをビルドしてデプロイするためのワークフローなど、必要なリソースが含まれています。このチュートリアルでは、モダンな 3 層ウェブアプリケーションブループリントを使用して Amazon でプロジェクトを作成する手順を説明します。 CodeCatalystこのチュートリアルには、デプロイされたサンプルを確認したり、他のユーザーに作業してもらうように依頼したり、AWS アカウント プルリクエストがマージされると自動的にビルドされて接続中のリソースにデプロイされるプルリクエストを使ってコードを変更したりすることも含まれます。レポート、アクティビティフィード、その他のツールを使ってプロジェクトを作成すると、 AWS AWS アカウント ブループリントはプロジェクトに関連するリソースを作成します。 CodeCatalyst ブループリントファイルを使用すると、サンプルモダンアプリケーションを構築してテストし、内のインフラストラクチャにデプロイできます。 AWS クラウド

次の図は、 CodeCatalyst 内のツールを使用して課題を追跡し、マージして自動的に作成し、CodeCatalyst AWS CDK AWS CloudFormation インフラストラクチャを許可およびプロビジョニングするアクションを実行するプロジェクト内のワークフローを開始する方法を示しています。

AWS アカウント アクションは関連するリソースを生成し、API Gateway AWS Lambdaエンドポイントを使用してアプリケーションをサーバーレス関数にデプロイします。 AWS Cloud Development Kit (AWS CDK) このアクションは 1 AWS CDK AWS CloudFormation つ以上のスタックをテンプレートに変換し、スタックをにデプロイします。 AWS アカウントスタック内のリソースには、動的ウェブコンテンツを配信するための Amazon CloudFront リソース、アプリケーションデータ用の Amazon DynamoDB インスタンス、デプロイされたアプリケーションをサポートするロールとポリシーが含まれます。



モダン 3 層ウェブアプリケーションブループリントを使用してプロジェクトを作成すると、プロジェクトは次のリソースで作成されます。

プロジェクト内: CodeCatalyst

- サンプルコードとワークフロー YAML を含むソースリポジトリ
- デフォルトブランチに変更が加えられるたびにサンプルコードをビルドしてデプロイするワークフロー。
- 作業の計画と追跡に使用できる課題ボードとバックログ
- サンプルコードに含まれる自動レポートを含むテストレポートスイート

関連する AWS アカウント:

- アプリケーションに必要なリソースを作成する 3 AWS CloudFormation つのスタック。

このチュートリアルで、 CodeCatalyst またこのチュートリアルの一部として作成されるリソースの詳細については、を参照してくださいリファレンス。 AWS

> **Note**
>
> プロジェクトに含まれるリソースとサンプルは、選択するブループリントによって異なります。Amazon CodeCatalyst では、定義された言語またはフレームワークに関連するリソースを定義するプロジェクトブループリントをいくつか提供しています。ブループリントの詳細については、を参照してください。 CodeCatalyst ブループリントを使用した包括的なプロジェクトの作成

トピック

前提条件

このチュートリアルでモダンアプリケーションプロジェクトを作成するには、以下のタスクを完了している必要があります。のセットアップと へのサインイン CodeCatalyst

- サインイン用の AWS Builder ID CodeCatalyst を用意してください。
- スペースに所属していて、そのスペースのスペース管理者またはパワーユーザーロールが割り当てられていること。詳細については、スペースの作成、ユーザーにスペース許可を付与する、およびスペース管理者ロールを参照してください。

- AWS アカウント 自分のスペースに関連付けて、サインアップ時に作成した IAM ロールを持ってください。たとえば、サインアップ時に、ロールポリシーと呼ばれるロールポリシーを使用してサービスロールを作成することを選択できます。CodeCatalystWorkflowDevelopmentRole-*spaceName*CodeCatalystWorkflowDevelopmentRole-ルには一意の識別子が付加された名前が付きます。ロールとロールポリシーの詳細については、を参照してくださいCodeCatalystWorkflowDevelopmentRole-*spaceName* サービスロールについて。ロールを作成する手順については、を参照してくださいアカウントとスペースのCodeCatalystWorkflowDevelopmentRole-*spaceName*ロールの作成。

ステップ 1: 最新の 3 層 Web アプリケーションプロジェクトを作成する

プロジェクトを作成したら、コードの開発とテスト、開発タスクの調整、プロジェクト指標の表示を行います。プロジェクトには開発ツールやリソースも含まれています。

このチュートリアルでは、モダンな 3 層ウェブアプリケーションブループリントを使用してインタラクティブなアプリケーションを作成します。プロジェクトの一部として自動的に作成され実行されるワークフローが、アプリケーションをビルドしてデプロイします。ワークフローは、スペースのすべてのロールとアカウント情報が設定されて初めて正常に実行されます。ワークフローが正常に実行されたら、エンドポイント URL にアクセスしてアプリケーションを確認できます。

ブループリントを使用してプロジェクトを作成するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. CodeCatalyst コンソールで、プロジェクトを作成したいスペースに移動します。
3. [プロジェクトを作成] を選択します。
4. [設計図から始める] を選択します。
5. [Search] バーに「**modern**」と入力します。
6. モダン 3 層 Web アプリケーションブループリントを選択し、[次へ] を選択します。
7. [プロジェクトに名前を付ける] に、プロジェクト名を入力します。例:

   MyExampleProject.

   > **Note**
   >
   > 名前はスペース内で一意でなければなりません。

8. 「アカウント」で、 AWS アカウント サインアップ時に追加した名前を選択します。ブループリントは、このアカウントにリソースをインストールします。
9. [Deployment Role] で、サインアップ時に追加したロールを選択します。例えば、[CodeCatalystWorkflowDevelopmentRole-*spaceName*] を選択します。

   ロールがリストにない場合は、ロールを追加してください。ロールを追加するには、[Add IAM role] を選択し、 AWS アカウントそのロールを自分に追加します。詳細については、「接続された AWS リソースへのアクセスを許可する AWS アカウント」を参照してください。
10. コンピュートプラットフォームで Lambda を選択します。
11. 「フロントエンドホスティングオプション」で、「Amplify Hosting」を選択します。詳細については AWS Amplify、「ホスティングとは AWS Amplify」を参照してください。 『AWS Amplify ユーザーガイド』の。
12. [デプロイリージョン] に、Mysfits AWS リージョン アプリケーションとサポートリソースをブループリントでデプロイしたい場所のリージョンコードを入力します。リージョンコードのリストについては、の「リージョナルエンドポイント」を参照してください。AWS 全般のリファレンス
13. [アプリケーション名] はデフォルトのままにします。mysfits*string*
14. (オプション) [プロジェクトプレビューを生成] で [コードを表示] を選択し、ブループリントがインストールするソースファイルをプレビューします。[ワークフローを表示] を選択して、ブループリントがインストールする CI/CD ワークフロー定義ファイルをプレビューします。プレビューは選択内容に基づいて動的に更新されます。
15. [プロジェクトを作成] を選択します。

プロジェクトワークフローは、プロジェクトを作成するとすぐに開始されます。コードのビルドとデプロイが完了するまでには少し時間がかかります。それまでの間、他の人をプロジェクトに招待してください。

ステップ 2: プロジェクトに誰かを招待する

プロジェクトを設定したら、他の人を招待して一緒に仕事をしてもらいます。

プロジェクトに誰かを招待するには

1. ユーザーを招待したいプロジェクトに移動します。
2. ナビゲーションペインで [プロジェクト設定] を選択します。
3. 「メンバー」タブで「招待」を選択します。

4. プロジェクトのユーザーとして招待したい人のメールアドレスを入力します。複数のメールアドレスをスペースまたはカンマで区切って入力できます。プロジェクトのメンバーではないスペースのメンバーの中から選ぶこともできます。
5. ユーザーのロールを選択します。

   ユーザーを追加し終えたら、[招待] を選択します。

ステップ 3: 課題を作成して共同作業を行い、作業を追跡する

CodeCatalyst プロジェクトに関係する機能、タスク、バグ、その他の課題の追跡に役立ちます。課題を作成して、必要な作業やアイデアを追跡できます。デフォルトでは、課題を作成するとバックログに追加されます。課題をボードに移動して進行中の作業を追跡できます。特定のプロジェクトメンバーに課題を割り当てることもできます。

プロジェクトの課題を作成するには

1. ナビゲーションペインで [Issues] を選択します。
2. [課題の作成] を選択します。
3. [課題のタイトル] に、課題の名前を入力します。必要に応じて、問題の説明を入力します。この例では、**`make a change in the src/mysfit_data.json file.`**
4. 優先度、見積もり、ステータス、ラベルを選択します。「担当者」で「+Add me」を選択し、課題を自分に割り当てます。
5. [課題を作成] を選択します。これで、課題がボードに表示されるようになりました。カードを選択して、課題を [進行中] 列に移動します。

詳細については、「で問題のある作業を追跡して整理する CodeCatalyst」を参照してください。

ステップ 4: ソースリポジトリを表示する

ブループリントは、アプリケーションやサービスを定義しサポートするファイルを含むソースリポジトリをインストールします。ソースリポジトリの注目すべきディレクトリとファイルは次のとおりです。

- .cloud9 ディレクトリ — 開発環境のサポートファイルが含まれています。 AWS Cloud9
- .codecatalyst ディレクトリ — YAML ブループリントに含まれる各ワークフローのワークフロー定義ファイルが含まれます。

- .idea ディレクトリ — 開発環境のサポートファイルが含まれます。 JetBrains
- .vscode ディレクトリ — Visual Studio Code 開発環境のサポートファイルが含まれています。
- CDKStacks ディレクトリ — AWS CDK 内のインフラストラクチャを定義するスタックファイルが含まれます。 AWS クラウド
- src ディレクトリ — アプリケーションのソースコードが含まれます。
- tests ディレクトリ — アプリケーションのビルドとテスト時に実行される自動 CI/CD ワークフローの一部として実行されるインテグテストとユニットテスト用のファイルが含まれます。
- Web ディレクトリ — フロントエンドのソースコードが含まれます。その他のファイルには、`package.json`プロジェクトに関する重要なメタデータを含むファイル、Web `index.html` サイトのページ、リンティングコード用のファイル、`.eslintrc.cjstsconfig.json`ルートファイルやコンパイラオプションを指定するファイルなどのプロジェクトファイルが含まれます。
- `Dockerfile`ファイル — アプリケーションのコンテナを記述します。
- `README.md`file — プロジェクトの設定情報が含まれます。

プロジェクトのソースリポジトリに移動するには

1. プロジェクトに移動し、次のいずれかを実行します。
   - プロジェクトの概要ページで、一覧から目的のリポジトリを選択し、[リポジトリを表示] を選択します。
   - ナビゲーションペインで [コード] を選択してから、[ソースリポジトリ] を選択します。「ソースリポジトリ」で、リストからリポジトリの名前を選択します。フィルターバーにリポジトリ名の一部を入力することで、リポジトリのリストをフィルタリングできます。
2. リポジトリのホームページには、リポジトリの内容と、プルリクエストの数やワークフローなどの関連リソースに関する情報が表示されます。デフォルトでは、デフォルトブランチのコンテンツが表示されます。ドロップダウンリストから別のブランチを選択することでビューを変更できます。

ステップ 5: テストブランチを含む開発環境を作成し、コードをすばやく変更します。

Dev Environment を作成すれば、ソースリポジトリ内のコードをすばやく編集できます。このチュートリアルでは、以下のことを行うことを前提としています。

- AWS Cloud9 開発環境を作成する。

- 開発環境を作成するときに、メインブランチ以外の新しいブランチで作業するオプションを選択してください。
- `test`この新しいブランチにはその名前を使用してください。

後のステップでは、Dev Environment を使用してコードを変更し、プルリクエストを作成します。

新しいブランチで開発環境を作成するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. 開発環境を作成するプロジェクトに移動します。
3. プロジェクトのソースリポジトリのリストからリポジトリの名前を選択します。または、ナビゲーションペインで [コード] を選択し、[ソースリポジトリ] を選択して、開発環境を作成するリポジトリを選択します。
4. リポジトリのホームページで、「開発環境を作成」を選択します。
5. ドロップダウンメニューからサポートされている IDE を選択します。詳細については、「開発環境でサポートされる統合開発環境」を参照してください。
6. クローンするリポジトリを選択し、[新しいブランチで作業する] を選択し、[ブランチ名] フィールドにブランチ名を入力し、[ブランチの作成元] ドロップダウンメニューから新しいブランチを作成するブランチを選択します。
7. オプションで、開発環境のエイリアスを追加します。
8. オプションで、[開発環境設定] 編集ボタンを選択して、開発環境のコンピューティング、ストレージ、またはタイムアウト設定を編集します。
9. [作成] を選択します。開発環境の作成中は、開発環境のステータス列に [開始中] と表示され、開発環境が作成されると、ステータス列に [実行中] と表示されます。選択した IDE の開発環境を含む新しいタブが開きます。コードを編集し、変更をコミットしてプッシュできます。

このセクションでは、プルリクエストがマージされると自動的にビルドされ、 CodeCatalyst AWS アカウント 接続中のリソースにデプロイされるプルリクエストを含むコードに変更を加えることで、生成されたサンプルアプリケーションを操作します。

`src/mysfit_data.json`ファイルに変更を加えるには

1. プロジェクトの Dev Environment に移動します。で AWS Cloud9、サイド・ナビゲーション・メニューを展開してファイルをブラウズします。`mysfitssrc`、を展開して開きます`src/mysfit_data.json`。

2. ファイルで、"Age":フィールドの値を 6 から 12 に変更します。行は次のようになるはずです。

```
    {
        "Age": 12,
        "Description": "Twilight's personality sparkles like the night sky and is
 looking for a forever home with a Greek hero or God. While on the smaller side
 at 14 hands, he is quite adept at accepting riders and can fly to 15,000 feet.
 Twilight needs a large area to run around in and will need to be registered with
 the FAA if you plan to fly him above 500 feet. His favorite activities include
 playing with chimeras, going on epic adventures into battle, and playing with a
 large inflatable ball around the paddock. If you bring him home, he'll quickly
 become your favorite little Pegasus.",
        "GoodEvil": "Good",
        "LawChaos": "Lawful",
        "Name": "Twilight Glitter",
        "ProfileImageUri": "https://www.mythicalmysfits.com/images/
pegasus_hover.png",
        "Species": "Pegasus",
        "ThumbImageUri": "https://www.mythicalmysfits.com/images/pegasus_thumb.png"
    },
```

3. ファイルを保存します。
4. **cd /projects/mysfits**コマンドで mysfits リポジトリに移動します。
5. git add、git commit、および git push コマンドを使用して、変更を追加、コミット、プッシュします。

```
git add .
git commit -m "make an example change"
git push
```

ステップ 6: 最新のアプリケーションを構築するワークフローを表示する

モダンアプリケーションプロジェクトを作成したら、 CodeCatalyst ワークフローを含むいくつかのリソースをユーザーに代わって生成します。ワークフローは、コードのビルド、テスト、デプロイの方法を説明する.yaml ファイルで定義される自動化された手順です。

このチュートリアルでは、 CodeCatalyst ワークフローを作成し、プロジェクトの作成時に自動的に開始しました。(プロジェクトを作成してからどのくらい経ったかによっては、ワークフローがまだ

実行されている場合があります)。次の手順を使用してワークフローの進行状況を確認し、生成されたログとテストレポートを確認し、最後にデプロイされたアプリケーションの URL に移動します。

ワークフローの進行状況を確認するには

1. CodeCatalyst コンソールのナビゲーションペインで [CI/CD] を選択し、[ワークフロー] を選択します。

   ワークフローのリストが表示されます。これらは、 CodeCatalyst プロジェクトを作成したときにブループリントが生成して開始したワークフローです。
2. ワークフローのリストを確認してください。次の 4 つが表示されるはずです。

   - 上部の 2 つのワークフローは、`test`前に作成したブランチに対応しています[ステップ 5: テストブランチを含む開発環境を作成し、コードをすばやく変更します。](#)。`main`これらのワークフローはブランチ上のワークフローのクローンです。ApplicationDeploymentPipeline`main`はブランチで使用するように設定されているためアクティブではありません。プルリクエストが行われていないため、OnPullRequestワークフローは実行されませんでした。
   - 一番下の 2 つのワークフローは、`main`ブループリントを先に実行したときに作成されたブランチに対応しています。ApplicationDeploymentPipelineワークフローはアクティブで、実行中 (または完了) しています。

   > **Note**
   >
   > ApplicationDeploymentPipeline実行が Build @cdk_bootstrap DeployBackendまたはエラーで失敗した場合は、以前に Modern 3 層 Web アプリケーションを実行していて、そのために古いリソースが残され、現在のブループリントとの競合が発生したことが原因である可能性があります。これらの古いリソースを削除してから、ワークフローを再実行する必要があります。詳細については、「[リソースをクリーンアップする](#)」を参照してください。
3. ApplicationDeploymentPipeline`main`下部にあるブランチに関連するワークフローを選択します。`main`このワークフローはブランチのソースコードを使用して実行されました。

   ワークフロー図が表示されます。この図には複数のブロックが示されており、それぞれがタスクまたはアクションを表しています。ほとんどのアクションは縦に並んでいて、一番上のアクションが下のアクションより先に実行されます。並べて配置されたアクションはparallel 実行されます。グループ化されたアクションをすべて正常に実行しないと、その下のアクションが開始されません。

主なブロックは以下のとおりです。

- WorkflowSource— このブロックはソースリポジトリを表します。他の情報の中でも、ソースリポジトリ名 (mysfits) と、ワークフローの実行を自動的に開始したコミットが表示されます。 CodeCatalyst このコミットは、プロジェクトを作成したときに生成されました。
- ビルド — このブロックは 2 つのアクションをグループ化したもので、次のアクションを開始するためには両方とも正常に完了する必要があります。
- DeployBackend— このブロックは、アプリケーションのバックエンドコンポーネントをクラウドにデプロイするアクションを表します。 AWS
- テスト — このブロックは 2 つのテストアクションをグループ化したもので、両方とも正常に完了しないと次のアクションが開始されません。
- DeployFrontend— このブロックは、アプリケーションのフロントエンドコンポーネントをクラウドにデプロイするアクションを表します。 AWS

4. 「定義」タブ (上部付近) を選択します。ワークフロー定義ファイルが右側に表示されます。このファイルには以下の注目すべきセクションがあります。

   - `Triggers`上部にセクションがあります。このセクションは、`main`ソースリポジトリのブランチにコードがプッシュされるたびにワークフローを開始する必要があることを示しています。他のブランチ (など`test`) にプッシュしても、このワークフローは開始されません。`main`ワークフローはブランチ上のファイルを使用して実行されます。
   - `Actions`のセクション`Triggers`。このセクションでは、ワークフロー図に表示されるアクションを定義します。

5. 「最新の状態」タブ (上部近く) を選択し、ワークフロー図内の任意のアクションを選択します。
6. 右側の「構成」タブを選択すると、前回の実行時にアクションが使用した構成設定が表示されます。ワークフロー定義ファイルには、各構成設定に対応するプロパティがあります。
7. コンソールは開いたままにして、次の手順に進みます。

ビルドログとテストレポートを確認するには

1. 「最新の状態」タブを選択します。
2. ワークフロー図で、DeployFrontendアクションを選択します。
3. アクションが終了するまでお待ちください。「進行中」アイコン
   ( )

が「成功」アイコン
(✓)
に変わるのを待ってください。

4. build_backend アクションを選択します。
5. Logs タブを選択し、いくつかのセクションを展開すると、これらのステップのログメッセージが表示されます。バックエンド設定に関連するメッセージを表示できます。
6. 「レポート」タブを選択し、`backend-coverage.xml`レポートを選択します。 CodeCatalyst 関連するレポートが表示されます。レポートには、実行されたコードカバレッジテストと、テストによって正常に検証されたコード行の割合 (80% など) が表示されます。

   テストレポートの詳細については、を参照してくださいワークフローを使用したテスト。

   > **Tip**
   >
   > ナビゲーションペインで [Reports] を選択してテストレポートを表示することもできます。

7. CodeCatalyst コンソールは開いたままにして、次の手順に進みます。

モダンアプリケーションが正常にデプロイされたことを確認するには

1. ApplicationDeploymentPipelineワークフローに戻り、最新の実行の Run-*string* リンクを選択します。
2. DeployFrontendワークフロー図でアクションを見つけて、「アプリを表示」リンクを選択します。Mysfit ウェブサイトが表示されます。

   > **Note**
   >
   > DeployFrontendアクション内に [アプリを表示] リンクが表示されない場合は、必ず run ID リンクを選択してください。

3. トワイライトグリッターという名前のペガサス Mysfit を探してください。年齢の値を書き留めてください。そうです6。コードを変更して年齢を更新します。

ステップ 7: 他の人に変更内容を確認してもらう

という名前のブランチに変更が加えられたらtest、プルリクエストを作成して他の人にレビューを依頼できます。testブランチの変更をブランチにマージするプルリクエストを作成するには、次の手順を実行します。main

プルリクエストを作成するには:

1. プロジェクトに移動します。
2. 次のいずれかを行います。
   - ナビゲーションペインで [コード] を選択し、[プルリクエスト] を選択し、[プルリクエストの作成] を選択します。
   - リポジトリのホームページで [More] を選択し、[Create pull request (プルリクエストの作成)] を選択します。
   - プロジェクトページで [プルリクエストを作成] を選択します。
3. [ソースリポジトリ] で、指定したソースリポジトリがコミットされたコードを含むソースリポジトリであることを確認します。このオプションは、リポジトリのメインページからプルリクエストを作成しなかった場合にのみ表示されます。
4. 「宛先ブランチ」で、レビュー後にコードをマージするブランチを選択します。
5. 「ソースブランチ」で、コミットされたコードを含むブランチを選択します。
6. プルリクエストのタイトルに、レビューが必要な内容とその理由を他のユーザーが理解しやすいタイトルを入力します。
7. (オプション) プルリクエストの説明には、課題へのリンクや変更内容の説明などの情報を入力します。

   > **Tip**
   >
   > 「説明を書く」を選択すると、 CodeCatalyst プルリクエストに含まれる変更の説明が自動的に生成されます。自動生成された説明は、プルリクエストに追加した後で変更できます。
   > この機能を使用するには、そのスペースでジェネレーティブ AI 機能を有効にする必要があります。詳細については、「ジェネレーティブ AI 機能の管理」を参照してください。

8. (オプション) [課題] で [課題をリンク] を選択し、一覧から課題を選択するか、ID を入力します。課題をリンク解除するには、リンク解除アイコンを選択します。

9. (オプション)「必須のレビュー担当者を追加」で、「必須のレビュー担当者を追加」を選択します。プロジェクトメンバーのリストから選択して追加します。プルリクエストをターゲットブランチにマージする前に、必須のレビュー担当者が変更を承認する必要があります。

> **Note**
>
> レビュー担当者を必須レビュー担当者とオプションレビュー担当者の両方として追加することはできません。自分をレビュー担当者として追加することはできません。

10. (オプション)「任意のレビュー担当者を追加」で、「任意のレビュー担当者を追加」を選択します。プロジェクトメンバーのリストから選択して追加します。プルリクエストをターゲットブランチにマージする前に、任意のレビュアーが変更を要件として承認する必要はありません。
11. ブランチ間の違いを確認してください。プルリクエストに表示される違いは、ソースブランチのリビジョンと、プルリクエストが作成された時点でのターゲットブランチのヘッドコミットであるマージベースのリビジョンとの間の変更です。変更が表示されない場合は、ブランチが同じか、ソースとターゲットの両方に同じブランチを選択した可能性があります。
12. プルリクエストにレビューしたいコードと変更が含まれていることを確認したら、[Create] を選択します。

> **Note**
>
> プルリクエストを作成したら、コメントを追加できます。コメントは、プルリクエストやファイル内の個々の行だけでなく、プルリクエスト全体にも追加できます。@ 記号に続けてファイル名を付けると、ファイルなどのリソースへのリンクを追加できます。

プルリクエストを作成すると、OnPullRequest`test`ワークフローはブランチ内のソースファイルの使用を開始します。レビュー担当者がコードの変更を承認している間は、ワークフローを選択してテスト出力を見ることで結果を確認できます。

変更を確認したら、コードをマージできます。コードをデフォルトブランチにマージすると、変更内容をビルドしてデプロイするワークフローが自動的に開始されます。

コンソールからのプルリクエストをマージするには CodeCatalyst

1. モダンアプリケーションプロジェクトに移動します。

2. プロジェクトページの「オープンプルリクエスト」で、マージするプルリクエストを選択します。プルリクエストが表示されない場合は、[View all] を選択し、一覧からプルリクエストを選択します。[Merge (マージ)] を選択します。
3. プルリクエストに使用可能なマージストラテジーから選択します。オプションで、プルリクエストをマージした後にソースブランチを削除するオプションを選択または選択解除し、[マージ] を選択します。

> **Note**
>
> 「マージ」ボタンがアクティブになっていないか、「マージ不可」というラベルが表示されている場合は、1 人以上の必須レビュアーがプルリクエストをまだ承認していないか、プルリクエストをコンソールでマージできません。 CodeCatalyst プルリクエストを承認していないレビュアーは、プルリクエストの詳細領域の「概要」に時計アイコンで示されます。必要なレビュアーが全員プルリクエストを承認したのに Merge ボタンがまだアクティブになっていない場合は、マージコンフリクトが発生している可能性があります。 CodeCatalyst宛先ブランチのマージコンフリクトをコンソールで解決してからプルリクエストをマージすることも、コンフリクトを解決してローカルでマージし、CodeCatalystマージを含むコミットをにプッシュすることもできます。詳細については、プルリクエストのマージ (Git)および Git のマニュアルを参照してください。

testブランチの変更をブランチにマージすると、ApplicationDeploymentPipeline変更内容をビルドしてデプロイするワークフローが自動的に開始されます。**main**

ApplicationDeploymentPipeline マージされたコミットがワークフロー全体で実行されるのを確認するには

1. ナビゲーションペインで [CI/CD] を選択し、[ワークフロー] を選択します。
2. [ワークフロー] の [ApplicationDeploymentPipeline最近の実行] を展開します。マージコミットによって開始されたワークフローの実行を確認できます。オプションでこれを選択して実行の進行状況を確認することもできます。
3. 実行が完了したら、前にアクセスした URL をリロードします。ペガサスを表示して、年齢が変わったことを確認します。

ステップ 8: 課題をクローズする

問題が解決したら、 CodeCatalyst コンソールでクローズできます。

プロジェクトの課題をクローズするには

1. プロジェクトに移動します。
2. ナビゲーションペインで [Issues] を選択します。
3. rag-and-drop 課題を [完了] 列に追加します。

詳細については、「で問題のある作業を追跡して整理する CodeCatalyst」を参照してください。

リソースをクリーンアップする

CodeCatalyst AWS このチュートリアルの痕跡をクリーンアップして、ご使用の環境から削除してください。

このチュートリアルで使用したプロジェクトを引き続き使用するか、プロジェクトとそれに関連するリソースを削除するかを選択できます。

> **Note**
>
> このプロジェクトを削除すると、そのプロジェクトのすべてのメンバーのリポジトリ、イシュー、アーティファクトが削除されます。

プロジェクトを削除するには

1. プロジェクトに移動し、[プロジェクト設定] を選択します。
2. [General] (全般) タブを選択します。
3. プロジェクト名の下にある [プロジェクトを削除] を選択します。

AWS CloudFormation と Amazon S3 のリソースを削除するには

1. AWS Management Console CodeCatalystスペースに追加したのと同じアカウントでにサインインします。
2. AWS CloudFormationサービスにアクセスします。
3. mysfits ###############。
4. 開発用 mysfits ストリングスタックを削除します#
5. CDKToolkit スタックを選択します (ただし、削除はしないでください)。[リソース] タブを選択します。StagingBucketリンクを選択し、Amazon S3 のバケットとバケットの内容を削除します。

   > **Note**
   >
   > このバケットを手動で削除しないと、Modern 3 層ウェブアプリケーションブループリントを再実行したときにエラーが表示されることがあります。

6. (オプション) CDKToolkit スタックを削除します。

リファレンス

モダンな 3 層ウェブアプリケーションブループリントは、 CodeCatalyst AWS クラウド内のスペースとアカウントにリソースをデプロイします。 AWS これらのリソースは以下のとおりです。

- CodeCatalyst あなたのスペースでは:

- CodeCatalyst 以下のリソースを含むプロジェクト:
  - ソースリポジトリ — このリポジトリには、「Mysfits」Web アプリケーションのサンプルコードが含まれています。
  - ワークフロー — このワークフローは、デフォルトブランチに変更が加えられるたびに Mysfits アプリケーションコードをビルドしてデプロイします。
  - 課題ボードとバックログ — このボードとバックログは、作業の計画と追跡に使用できます。
  - テストレポートスイート — このスイートには、サンプルコードに含まれる自動レポートが含まれています。
- 関連する AWS アカウント:
  - CDKToolkit スタック — このスタックは以下のリソースをデプロイします。
    - Amazon S3 ステージングバケット、バケットポリシー、 AWS KMS およびバケットの暗号化に使用されるキー。
    - デプロイアクションの IAM デプロイロール。
    - AWS スタック内のリソースをサポートする IAM ロールとポリシー。

    > **Note**
    >
    > CDKToolkit はデプロイごとに解体して再作成されることはありません。これは、をサポートするために各アカウントで開始されるスタックです。 AWS CDK
  - development-mysfits ***BackEnd#######*** — このスタックは以下のバックエンドリソースをデプロイします。
    - Amazon API Gateway のエンドポイント。
    - AWS スタック内のリソースをサポートする IAM ロールとポリシー。
    - AWS Lambda 関数とレイヤーは、最新のアプリケーション用のサーバーレスコンピューティングプラットフォームを提供します。
    - バケットデプロイと Lambda 関数のための IAM ポリシーとロール。
  - mysfits ***#######*** — このスタックはフロントエンドアプリケーションをデプロイします。 AWS Amplify

以下も参照してください。

AWS このチュートリアルの一部としてリソースが作成されるサービスの詳細については、以下を参照してください。

- Amazon S3 — 業界トップのスケーラビリティ、データの高可用性、セキュリティ、パフォーマンスを提供するオブジェクトストレージサービスにフロントエンドアセットを保存するサービス。詳細については、Amazon S3 ユーザーガイドを参照してください。
- Amazon API Gateway — REST、HTTP、API をあらゆる規模で作成、公開、保守、モニタリング、保護するためのサービスです。詳細については、「API Gateway 開発者ガイド」を参照してください。 WebSocket
- Amplify — フロントエンドアプリケーションをホストするためのサービス。詳細については、「AWS Amplify ホスティングユーザーガイド」を参照してください。
- AWS Cloud Development Kit (AWS CDK)— クラウドインフラストラクチャをコードで定義し、それを介してプロビジョニングするためのフレームワーク AWS CloudFormation。 AWS CDK には、 AWS CDK アプリやスタックを操作するためのコマンドラインツールである AWS CDK Toolkit が含まれています。詳細については、「AWS Cloud Development Kit (AWS CDK) デベロッパーガイド」を参照してください。
- Amazon DynamoDB — データを保存するためのフルマネージド型の NoSQL データベースサービスです。詳細については、Amazon DynamoDB開発者ガイドを参照してください。
- AWS Lambda— サーバーのプロビジョニングや管理を行わずに、高可用性コンピューティングインフラストラクチャでコードを呼び出すサービス。詳細については、「AWS Lambda デベロッパーガイド」を参照してください。
- AWS IAM — AWS およびそのリソースへのアクセスを安全に制御するためのサービス。詳細については、IAM ユーザーガイドを参照してください。

チュートリアル:空のプロジェクトから始めて、手動でリソースを追加する

プロジェクトの作成時に空のプロジェクトブループリントを選択すれば、あらかじめ定義されたリソースがない空のプロジェクトを作成できます。空のプロジェクトを作成したら、プロジェクトの必要に応じてリソースを作成して追加できます。ブループリントなしで作成されたプロジェクトは作成時には空になるため、 CodeCatalyst このオプションを開始するにはリソースの作成と設定に関するより多くの知識が必要です。

トピック

- コード変更をビルド、テスト、デプロイするためのワークフローを作成します。
- プロジェクトに誰かを招待してください。
- 課題を作成して、共同作業や作業の追跡を行います。

前提条件

空のプロジェクトを作成するには、スペース管理者またはパワーユーザーロールが割り当てられている必要があります。に初めてサインインする場合は CodeCatalyst、を参照してくださいのセットアップと へのサインイン CodeCatalyst。

空のプロジェクトを作成します。

プロジェクトを作成することは、共同作業を進めるための第一歩です。ソースリポジトリやワークフローなどの独自のリソースを作成したい場合は、空のプロジェクトから始めることができます。

空のプロジェクトを作成するには

1. プロジェクトを作成するスペースに移動します。
2. スペースダッシュボードで、[プロジェクトの作成] を選択します。
3. [最初から開始] を選択します。
4. [プロジェクトに名前を付ける] に、プロジェクトに割り当てる名前を入力します。名前はスペース内で一意でなければなりません。
5. [プロジェクトを作成] を選択します。

空のプロジェクトができたので、次のステップはソースリポジトリの作成です。

ソースリポジトリを作成します。

ソースリポジトリを作成して、プロジェクトのコードを保存し、共同編集してください。プロジェクトメンバーは、このリポジトリをローカルコンピューターに複製してコードを編集できます。あるいは、サポートされているサービスでホストされているリポジトリをリンクすることもできますが、このチュートリアルでは説明していません。詳細については、「ソースリポジトリのリンク」を参照してください。

ソースリポジトリを作成するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。

2. プロジェクトに移動します。
3. ナビゲーションペインで [コード] を選択してから、[ソースリポジトリ] を選択します。
4. [リポジトリの追加] を選択し、[リポジトリの作成] を選択します。
5. [リポジトリ名] で、リポジトリの名前を指定します。このガイドではを使用しますが*codecatalyst-source-repository*、別の名前を選択することもできます。リポジトリ名は、プロジェクト内で一意である必要があります。リポジトリ名の要件の詳細については、を参照してくださいのソースリポジトリのクォータ CodeCatalyst。
6. (オプション)「説明」に、プロジェクト内の他のユーザーがリポジトリの使用目的を理解しやすくなるようなリポジトリの説明を追加します。
7. (オプション) `.gitignore` プッシュする予定のコードの種類に対応するファイルを追加します。
8. [作成] を選択します。

> **Note**
>
> CodeCatalyst `README.md`作成時にリポジトリにファイルを追加します。 CodeCatalyst また、main という名前のデフォルトブランチにリポジトリの初期コミットを作成します。README.md ファイルは編集または削除できますが、デフォルトブランチを変更または削除することはできません。

Dev Environment を作成すれば、リポジトリにコードをすばやく追加できます。このチュートリアルでは、を使用して開発環境を作成し AWS Cloud9、開発環境の作成時にメインブランチからブランチを作成するオプションを選択することをお勧めします。**`test`**このブランチにはこの名前を使用しますが、必要に応じて別のブランチ名を入力することもできます。

新しいブランチで開発環境を作成するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. 開発環境を作成するプロジェクトに移動します。
3. プロジェクトのソースリポジトリのリストからリポジトリの名前を選択します。または、ナビゲーションペインで [コード] を選択し、[ソースリポジトリ] を選択して、開発環境を作成するリポジトリを選択します。
4. リポジトリのホームページで、「開発環境を作成」を選択します。

5. ドロップダウンメニューからサポートされている IDE を選択します。詳細については、「開発環境でサポートされる統合開発環境」を参照してください。
6. クローンするリポジトリを選択し、[新しいブランチで作業する] を選択し、[ブランチ名] フィールドにブランチ名を入力し、[ブランチの作成元] ドロップダウンメニューから新しいブランチを作成するブランチを選択します。
7. オプションで、開発環境のエイリアスを追加します。
8. オプションで、[開発環境設定] 編集ボタンを選択して、開発環境のコンピューティング、ストレージ、またはタイムアウト設定を編集します。
9. [作成] を選択します。開発環境の作成中は、開発環境のステータス列に [開始中] と表示され、開発環境が作成されると、ステータス列に [実行中] と表示されます。選択した IDE の開発環境を含む新しいタブが開きます。コードを編集し、変更をコミットしてプッシュできます。

コード変更をビルド、テスト、デプロイするためのワークフローを作成します。

では CodeCatalyst、アプリケーションやサービスの構築、テスト、デプロイをワークフローで整理します。ワークフローはアクションで構成され、コードプッシュやプルリクエストのオープンや更新など、指定されたソースリポジトリイベントが発生した後に自動的に実行されるように設定できます。ワークフローの詳細については、「でワークフローを使用して構築、テスト、デプロイする CodeCatalyst」を参照してください。

ワークフローの開始方法の指示に従って、最初のワークフローを作成してください。

プロジェクトに誰かを招待してください。

カスタムプロジェクトを設定したら、他の人を招待して一緒に仕事をしてもらいます。

プロジェクトに誰かを招待するには

1. ユーザーを招待したいプロジェクトに移動します。
2. ナビゲーションペインで [プロジェクト設定] を選択します。
3. 「メンバー」タブで「招待」を選択します。
4. プロジェクトのユーザーとして招待したい人のメールアドレスを入力します。複数のメールアドレスをスペースまたはカンマで区切って入力できます。プロジェクトのメンバーではないスペースのメンバーの中から選ぶこともできます。
5. ユーザーのロールを選択します。

ユーザーを追加し終えたら、[招待] を選択します。

課題を作成して、共同作業や作業の追跡を行います。

CodeCatalyst プロジェクトに関係する機能、タスク、バグ、その他の課題の追跡に役立ちます。課題を作成して、必要な作業やアイデアを追跡できます。デフォルトでは、課題を作成するとバックログに追加されます。課題をボードに移動して進行中の作業を追跡できます。特定のプロジェクトメンバーに課題を割り当てることもできます。

プロジェクトの課題を作成するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。

   課題を作成したいプロジェクト内を移動していることを確認してください。すべてのプロジェクトを表示するには、ナビゲーションペインで [Amazon] を選択し CodeCatalyst、必要に応じて [すべてのプロジェクトを表示] を選択します。課題を作成または処理したいプロジェクトを選択します。
2. ナビゲーションペインで [Track] を選択し、[Backlog] を選択します。
3. [課題の作成] を選択します。
4. [課題のタイトル] に、課題の名前を入力します。必要に応じて、問題の説明を入力します。必要に応じて、問題のステータス、優先度、見積もりを選択します。プロジェクトメンバーのリストからプロジェクトメンバーに課題を割り当てることもできます。

   > **Tip**
   >
   > Amazon Q に課題を割り当てて、Amazon Q に問題の解決を試してもらうこともできます。成功すると、プルリクエストが作成され、問題のステータスが [レビュー中] に変わり、コードのレビューとテストが可能になります。詳細については、「チュートリアル: CodeCatalyst 生成 AI 機能を使用して開発作業を高速化する」を参照してください。
   > この機能を使用するには、そのスペースでジェネレーティブ AI 機能を有効にする必要があります。詳細については、「ジェネレーティブ AI 機能の管理」を参照してください。
5. [保存] を選択します。

課題を作成したら、プロジェクトメンバーに課題を割り当て、見積もり、かんばんボードで追跡できます。詳細については、「で問題のある作業を追跡して整理する CodeCatalyst」を参照してください。

チュートリアル: CodeCatalyst 生成 AI 機能を使用して開発作業を高速化する

生成 AI 機能が有効になっているスペース CodeCatalyst に Amazon のプロジェクトとソースリポジトリがある場合は、これらの機能を使用してソフトウェア開発を高速化できます。デベロッパーは、多くの場合、タスクを実行する時間よりも多くのタスクを実行します。多くの場合、変更のレビューのためのプルリクエストを作成するときに、チームメイトにコードの変更を説明するのに時間をかけず、他のユーザーが変更を自己説明的に見つけることを期待します。プルリクエスト作成者とレビューワーには、プルリクエストに関するすべてのコメントを徹底的に検索して読み取る時間もありません。特にプルリクエストに複数のリビジョンがある場合、 はソフトウェア開発のために Amazon Q デベロッパーエージェントと CodeCatalyst 統合し、チームメンバーがタスクをより迅速に達成し、作業の最も重要な部分に集中する時間を増やすのに役立つ生成 AI 機能を提供します。

Amazon Q Developer は、生成 AI を活用した会話型アシスタントで、 AWS アプリケーションの理解、構築、拡張、運用に役立ちます。でのビルドを加速するために AWS、Amazon Q を強化するモデルには、より完全で実用的な、参照される回答を生成する高品質の AWS コンテンツが拡張されています。詳細については、「Amazon Q デベロッパーユーザーガイド」の「Amazon Q デベロッパーとは」を参照してください。

> **Note**
>
> Amazon Bedrock を搭載： AWS 自動不正使用検出を実装。「Write description for me」、「Create content summary」、「Assign issues to Amazon Q feature with Amazon Q Developer Agent for Software Development」は Amazon Bedrock 上に構築されているため、ユーザーは Amazon Bedrock に実装されているコントロールを最大限に活用して、安全性、セキュリティ、人工知能 (AI) の責任ある使用を強制できます。

このチュートリアルでは、 の生成 AI 機能を使用して、プルリクエストの作成時にブランチ間の変更を CodeCatalyst 要約したり、プルリクエストに残されたコメントを要約したりする方法について説明します。また、コードの変更や改善に関するアイデアに問題を作成し、Amazon Q に割り当てる方法についても説明します。Amazon Q に割り当てられた問題の作業の一環として、Amazon Q がタス

クを提案できるようにする方法と、問題の作業の一環として作成するタスクを割り当てて作業する方法について説明します。

前提条件

このチュートリアル CodeCatalyst の機能を使用するには、まず を完了し、次のリソースにアクセスできる必要があります。

- にサインインするための AWS Builder ID または Single Sign-On (SSO) ID を持っている CodeCatalyst。
- プロジェクトは、生成 AI 機能が有効になっているスペースにあります。詳細については、「生成 AI 機能の管理」を参照してください。
- そのスペースのプロジェクトには、寄稿者またはプロジェクト管理者ロールがあります。
- プロジェクトには、少なくとも 1 つのソースリポジトリが設定されています。リンクされたリポジトリはサポートされていません。
- 生成 AI によって作成された初期ソリューションに問題を割り当てる場合、プロジェクトを Jira ソフトウェア拡張機能で設定することはできません。拡張機能は、この機能ではサポートされていません。

詳細については、「スペースの作成」、「で問題のある作業を追跡して整理する CodeCatalyst」、「で拡張機能を使用してプロジェクトに機能を追加する CodeCatalyst」、および「ユーザーロールによるアクセス許可の付与」を参照してください。

このチュートリアルは、Python で Modern 3 層ウェブアプリケーションブループリントを使用して作成されたプロジェクトに基づいています。別のブループリントで作成されたプロジェクトを使用する場合、ステップに従うことができますが、サンプルコードや言語など、いくつかの詳細が異なります。

プルリクエストの作成時にブランチ間のコード変更の概要を作成する

プルリクエストは、ユーザーや他のプロジェクトメンバーが、あるブランチから別のブランチへのコード変更を確認、コメント、マージできる主な方法です。プルリクエストを使用して、リリースされたソフトウェアのマイナーな変更や修正、主要な機能の追加、または新しいバージョンについて、コードの変更を共同で確認できます。プルリクエストの説明の一部としてコードの変更と変更の背後にあるインテントを要約することは、コードを確認する他のユーザーにとって役立ち、時間の経過とともにコードに加えられた変更の履歴を理解するのに役立ちます。ただし、デベロッパーは、レ

ビュー対象やコードの変更の背後にあるインテントを理解するのに十分な詳細で変更を記述するのではなく、コードに基づいて説明したり、あいまいな詳細を提供したりすることがよくあります。

Amazon Q にプルリクエストに含まれる変更の説明を作成させるプルリクエストを作成するときに、説明の書き込み機能を使用できます。このオプションを選択すると、Amazon Q はコード変更を含むソースブランチと、これらの変更をマージする送信先ブランチの違いを分析します。次に、それらの変更の概要と、それらの変更の意図と効果の最良の解釈を作成します。

> **Note**
>
> この機能は Git サブモジュールでは機能しません。プルリクエストの一部である Git サブモジュールの変更は要約されません。

この機能は、作成したプルリクエストで試すことができますが、このチュートリアルでは、Python ベースの Modern 3 層ウェブアプリケーション設計図で作成されたプロジェクトに含まれるコードに簡単な変更を加えてテストします。

> **Tip**
>
> 別のブループリントまたは独自のコードで作成されたプロジェクトを使用している場合でも、このチュートリアルに従うことができますが、このチュートリアルの例ではプロジェクトのコードと一致しません。以下の推奨例の代わりに、ブランチ内のプロジェクトのコードに簡単な変更を加え、次のステップに示すように、この機能をテストするプルリクエストを作成します。

まず、ソースリポジトリにブランチを作成します。次に、コンソールのテキストエディタを使用して、そのブランチのファイルに対して簡単なコード変更を行います。次に、プルリクエストを作成し、説明の書き込み機能を使用して、行った変更を要約します。

ブランチを作成するには (コンソール)

1. CodeCatalyst コンソールで、ソースリポジトリが存在するプロジェクトに移動します。
2. プロジェクトのソースリポジトリのリストからリポジトリの名前を選択します。または、ナビゲーションペインでコード を選択し、ソースリポジトリ を選択します。
3. ブランチを作成するリポジトリを選択します。
4. リポジトリの概要ページで、「詳細」を選択し、「ブランチの作成」を選択します。

5. ブランチの名前を入力します。
6. ブランチを作成するブランチを選択し、 の作成を選択します。

ブランチを作成したら、そのブランチ内のファイルを簡単な変更で編集します。この例では、test_endpoint.py ファイルを編集して、テストの再試行回数を から 5 **3** に変更します。

> **Tip**
>
> また、開発環境を作成または使用して、このコードを変更することもできます。詳細については、「開発環境の作成」を参照してください。

コンソールで**test_endpoint.py**ファイルを編集するには

1. **mysfits** ソースリポジトリの概要ページで、ブランチドロップダウンを選択し、前の手順で作成したブランチを選択します。
2. ファイル で、編集するファイルに移動します。例えば、 test_endpoint.py ファイルを編集するには、テスト を展開し、整数 を展開して、 を選択しますtest_endpoint.py。
3. [編集] を選択します。
4. 7 行目で、すべてのテストを再試行する回数を変更します。

   ```
   def test_list_all(retry=3):
   ```

   変更先:

   ```
   def test_list_all(retry=5):
   ```
5. コミット を選択し、変更をブランチにコミットします。

変更を含むブランチができたので、プルリクエストを作成できます。

変更の概要を含むプルリクエストを作成する

1. リポジトリの概要ページで、詳細 を選択し、プルリクエストの作成 を選択します。
2. 送信先ブランチ で、レビュー後にコードをマージするブランチを選択します。

> **Tip**
>
> この機能の最も簡単なデモンストレーションのために、前の手順でブランチを作成したブランチを選択します。例えば、リポジトリのデフォルトブランチからブランチを作成した場合は、プルリクエストの送信先ブランチとしてそのブランチを選択します。

3. ソースブランチ で、`test_endpoint.py`ファイルにコミットした変更を含むブランチを選択します。
4. プルリクエストのタイトル に、他のユーザーがレビューする必要がある内容と理由を理解するのに役立つタイトルを入力します。
5. プルリクエストの説明 で、「説明を記述する」を選択すると、Amazon Q がプルリクエストに含まれる変更の説明を作成します。
6. 変更の概要が表示されます。提案されたテキストを確認し、承認を選択して説明に追加します。
7. オプションで、コードに加えた変更をより適切に反映するように概要を変更します。このプルリクエストにレビューワーを追加するか、問題をリンクするかを選択することもできます。必要な追加変更が完了したら、 の作成 を選択します。

プルリクエストのコード変更に残っているコメントの概要を作成する

ユーザーがプルリクエストを確認すると、そのプルリクエストの変更について複数のコメントを残すことがよくあります。多くのレビューワーからのコメントが多数ある場合、フィードバックで共通のテーマを選択することは難しい場合があります。また、すべてのリビジョンですべてのコメントをレビューしたことを確認することもできます。コメント概要の作成機能を使用して、Amazon Q にプルリクエストのコード変更に残されたすべてのコメントを分析し、それらのコメントの概要を作成させることができます。

> **Note**
>
> コメントの概要は一時的なものです。プルリクエストを更新すると、概要は表示されなくなります。コンテンツの概要には、プルリクエスト全体に関するコメントは含まれず、プルリクエストのリビジョンのコードの違いに関するコメントだけが残ります。
> この機能は、Git サブモジュールのコード変更に関するコメントには対応しません。

プルリクエストでコメントの概要を作成するには

1. 前の手順で作成したプルリクエストに移動します。

   > **Tip**
   >
   > 必要に応じて、プロジェクトで任意のオープンプルリクエストを使用できます。ナビゲーションバーで、コード を選択し、プルリクエスト を選択し、開いているプルリクエストを選択します。

2. プルリクエストにコメントがまだない場合は、「変更」でプルリクエストにコメントを追加します。
3. 概要 で、コメント概要の作成 を選択します。完了すると、コメントの概要セクションが展開されます。
4. プルリクエストのリビジョンのコードの変更に関するコメントの概要を確認し、プルリクエストのコメントと比較します。

問題を作成して Amazon Q に割り当てる

開発チームは作業を追跡および管理するための問題を作成しますが、誰が作業すべきかが明確でない場合や、問題がコードベースの特定の部分を調査する必要があるか、その他の緊急作業に最初に参加する必要があるため、問題が長引くことがあります。ソフトウェア開発のために Amazon Q デベロッパーエージェントとの統合 CodeCatalyst が含まれます。Amazon Q という生成 AI アシスタントに問題を割り当てて、タイトルとその説明に基づいて問題を分析できます。問題を Amazon Q に割り当てると、評価のためのドラフトソリューションの作成が試みられます。これにより、Amazon Q はすぐに対処すべきリソースがない問題に対するソリューションに取り組み、お客様やお客様のチームが注意が必要な問題に集中して作業を最適化するのに役立ちます。

> **Tip**
>
> Amazon Q は、単純な問題と単純な問題に最適です。最良の結果を得るには、プレーン言語を使用して、実行したいことを明確に説明してください。

Amazon Q に問題を割り当てると、Amazon Q が問題にどのように対処するかを確認するまで、CodeCatalyst は問題をブロック済みとしてマークします。続行するには、次の 3 つの質問に答える必要があります。

- 実行するすべてのステップを確認するか、フィードバックなしで続行するか。各ステップを確認することを選択した場合は、Amazon Q が作成したアプローチに関するフィードバックで Amazon Q に返信して、必要に応じてそのアプローチを反復処理できます。このオプションを選択すると、Amazon Q は、ユーザーが作成したプルリクエストで残したフィードバックを確認することもできます。各ステップを確認しないことを選択した場合、Amazon Q は作業をより迅速に完了できますが、問題や作成したプルリクエストで提供したフィードバックは確認されません。
- ワークフローファイルを作業の一部として更新することを許可するかどうか。プロジェクトには、プルリクエストイベントの実行を開始するようにワークフローが設定されている場合があります。その場合は、ワークフロー YAML の作成または更新を含む Amazon Q が作成するプルリクエストによって、プルリクエストに含まれるワークフローの実行が開始される可能性があります。ベストプラクティスとして、作成したプルリクエストを確認して承認する前に、これらのワークフローを自動的に実行するワークフローがプロジェクトにないことを確認した場合を除き、Amazon Q によるワークフローファイルの作業を許可しないでください。
- 問題内の作業を Amazon Q 自体を含め、ユーザーに個別に割り当てることができる小さな増分に分割するタスクの作成を提案するかどうか。Amazon Q にタスクの提案と作成を許可すると、複数のユーザーが問題の個別の部分に取り組むことができるため、複雑な問題の開発を加速できます。また、各タスクを完了するために必要な作業が、それが属する問題よりも単純になるのが理想的であるため、作業全体を理解する複雑さを減らすこともできます。
- 使用するソースリポジトリ。プロジェクトに複数のソースリポジトリがある場合でも、Amazon Q は 1 つのソースリポジトリ内のコードでのみ動作します。リンクされたリポジトリはサポートされていません。

選択を行い、確認したら、Amazon Q は問題タイトルとその説明、および指定されたリポジトリ内のコードに基づいてリクエストが何であるかを判断しようとしている間、問題を進行中状態に移行します。ピン留めされたコメントが作成され、作業のステータスに関する更新が提供されます。データを確認した後、Amazon Q はソリューションに対する潜在的なアプローチを策定します。Amazon Q は、ピン留めされたコメントを更新し、すべての段階で問題の進行状況にコメントすることで、アクションを記録します。ピン留めされたコメントや返信とは異なり、作業の厳密な時系列記録は保持されません。むしろ、ピン留めされたコメントの最上位レベルに、その作業に関する最も関連性の高い情報を配置します。リポジトリに既に存在するコードのアプローチと分析に基づいてコードを作成しようとします。潜在的なソリューションが正常に生成されると、ブランチが作成され、そのブランチにコードがコミットされます。次に、そのブランチをデフォルトのブランチとマージするプルリクエストを作成します。Amazon Q が作業を完了すると、問題をレビュー中に移動し、評価できるコードがあることをユーザーとチームが認識できるようにします。

> **Note**
>
> この機能は、米国西部 (オレゴン) リージョンの問題でのみ使用できます。Jira ソフトウェア拡張機能で Jira を使用するようにプロジェクトを設定している場合は使用できません。さらに、ボードのレイアウトをカスタマイズしても、問題の状態は変更されない場合があります。最良の結果を得るには、標準ボードレイアウトのプロジェクトでのみこの機能を使用してください。
> この機能は Git サブモジュールでは機能しません。リポジトリに含まれる Git サブモジュールを変更することはできません。
> Amazon Q に問題を割り当てると、問題のタイトルや説明を変更したり、他のユーザーに割り当てたりすることはできません。問題から Amazon Q の割り当てを解除すると、現在のステップが終了し、作業が停止します。割り当てを解除すると、作業を再開したり、問題に再割り当てしたりすることはできません。
> ユーザーがタスクの作成を許可することを選択した場合、Amazon Q に割り当てられると、問題は自動的に「レビュー中」列に移動できます。ただし、「レビュー中」の問題には、「進行中」状態など、別の状態のタスクがまだ存在する可能性があります。

チュートリアルのこの部分では、Modern 3 層ウェブアプリケーション設計図で作成されたプロジェクトに含まれるコードの潜在的な機能に基づいて 3 つの問題を作成します。1 つは新しい mysfit クリーチャーを作成するための を追加し、もう 1 つはソート機能を追加し、もう 1 つは という名前のブランチを含むようにワークフローを更新するものです**`test`**。

> **Note**
>
> 異なるコードでプロジェクトで作業している場合は、そのコードベースに関連するタイトルと説明に関する問題を作成します。

問題を作成し、評価用のソリューションを生成するには

1. ナビゲーションペインで問題を選択し、ボードビューが表示されていることを確認します。
2. 問題の作成 を選択します。
3. 問題に、プレーン言語で何をするかを説明するタイトルを付けます。例えば、この問題の場合は、タイトルとして を入力します**`Create another mysfit named Quokkapus`**。説明 で、次の詳細を指定します。

```
Expand the table of mysfits to 13, and give the new mysfit the following
 characteristics:

Name: Quokkapus

Species: Quokka-Octopus hybrid

Good/Evil: Good

Lawful/Chaotic: Chaotic

Age: 216

Description: Australia is full of amazing marsupials, but there's nothing there
 quite like the Quokkapus.
She's always got a friendly smile on her face, especially when she's using her
 eight limbs to wrap you up
in a great big hug. She exists on a diet of code bugs and caffeine. If you've got
 some gnarly code that needsa
assistance, adopt Quokkapus and put her to work - she'll love it! Just make sure
 you leave enough room for
her to grow, and keep that coffee coming.
```

4. （オプション）問題に mysfit のサムネイルおよびプロファイル画像として使用するイメージをアタッチします。これを行う場合は、使用するイメージの詳細と理由を含むように説明を更新します。例えば、説明に「mysfit では、イメージファイルをウェブサイトにデプロイする必要があります。作業の一環として、この問題にアタッチされたこれらのイメージをソースリポジトリに追加し、イメージをウェブサイトにデプロイします。」

> **Note**
>
> アタッチされたイメージは、このチュートリアルのやり取り中にウェブサイトにデプロイされる場合とデプロイされない場合があります。イメージをウェブサイトに自分で追加し、プルリクエストの作成後に使用するイメージを指すように Amazon Q のコードを更新するためのコメントを残すことができます。

次のステップに進む前に、説明を確認し、必要なすべての詳細が含まれていることを確認してください。

5. 担当者 で、Amazon Q に割り当てる を選択します。

6. ソースリポジトリ で、プロジェクトコードを含むソースリポジトリを選択します。
7. 「Amazon Q が各ステップの後に停止する必要がある」をスライドし、必要に応じて作業セレクタのレビューをアクティブ状態に待機します。

> **Note**
>
> 各ステップの後に Amazon Q を停止するオプションを選択すると、問題または作成されたタスクにコメントして、コメントに基づいて Amazon Q のアプローチを最大 3 回変更するオプションを持つことができます。すべてのステップの後に Amazon Q を停止しないオプションを選択して作業を確認できるようにすると、Amazon Q はフィードバックを待っているわけではないが、コメントを残して Amazon Q が取る方向に影響を与えることができないため、作業が早く進む可能性があります。そのオプションを選択した場合、Amazon Q はプルリクエストに残されたコメントにも応答しません。

8. Amazon Q がワークフローファイルセレクタを非アクティブ状態のままにしておきます。
9. Amazon Q を許可 をスライドして、タスクセレクタの作成をアクティブ状態に提案します。
10. 問題の作成 を選択します。ビューが「問題」ボードに変更されます。
11. 「問題の作成」を選択して別の問題を作成します。今回は「」というタイトルで作成しますChange the get_all_mysfits() API to return mysfits sorted by the Age attribute。この問題を Amazon Q に割り当て、問題を作成します。
12. 「問題の作成」を選択して別の問題を作成します。今回は、タイトルが の問題を作成しますUpdate the OnPullRequest workflow to include a branch named test in its triggers。必要に応じて、説明のワークフローにリンクします。この問題は Amazon Q に割り当てますが、今回は Amazon Q にワークフローファイルの変更を許可するセレクタがアクティブ状態に設定されていることを確認します。問題を作成して、問題ボードに戻ります。

> **Tip**
>
> ワークフローファイルを含むファイルを検索するには、アットマーク (@) とファイル名を入力します。

問題を作成して割り当てると、問題は進行中 に移行します。Amazon Q は、問題内で進行状況を追跡するコメントをピン留めされたコメントに追加します。ソリューションへのアプローチを定義できる場合、問題の説明を、コードベースの分析を含む背景セクションと、ソリューションの作成に提案されたアプローチを詳述するアプローチセクションで更新します。Amazon Q が問題で説明されてい

る問題の解決策を考案することに成功すると、提案されたソリューションを実装するブランチとコードの変更が、そのブランチに作成されます。提案されたコードに Amazon Q が認識しているオープンソースコードとの類似性が含まれている場合、そのコードへのリンクを含むファイルが提供され、確認できるようになります。コードの準備ができたら、提案されたコードの変更を確認できるようにプルリクエストを作成し、そのプルリクエストへのリンクを問題に追加して、問題をレビュー中 に移動します。

> **Important**
>
> プルリクエストのコードの変更は、マージする前に必ず確認する必要があります。Amazon Q によって行われたコード変更をマージすると、他のコード変更と同様に、マージされたコードが適切にレビューされておらず、マージ時にエラーが含まれている場合、コードベースとインフラストラクチャコードに悪影響を及ぼす可能性があります。

Amazon Q によって行われた変更を含む問題とリンクされたプルリクエストを確認するには

1. 問題 で、進行中の Amazon Q に割り当てられた問題を選択します。コメントを確認して Amazon Q の進行状況をモニタリングします。存在する場合は、背景を確認し、問題の説明に記録します。Amazon Q がタスクを提案することを許可することを選択した場合は、提案されたタスクを確認し、必要なアクションを実行します。例えば、Amazon Q がタスクを提案し、順序を変更したり、特定のユーザーにタスクを割り当てたりする場合は、タスクの変更、追加、または順序の変更を選択し、必要な更新を実行します。問題の表示が完了したら、X を選択して問題ペインを閉じます。

   > **Tip**
   >
   > タスクの進行状況を表示するには、問題のタスクのリストからタスクを選択します。タスクはボードに個別の項目として表示されず、問題を通じてのみアクセスできます。タスクが Amazon Q に割り当てられている場合は、タスクを開いて、実行したいアクションを承認する必要があります。また、リンクされたプルリクエストは、タスク内でのみリンクとして表示されないため、タスクを開く必要があります。タスクから問題に戻るには、問題へのリンクを選択します。

2. 次に、「レビュー中」にある Amazon Q に割り当てられた問題を選択します。背景を確認し、問題の説明に記録するようにします。コメントを確認して、実行されたアクションを理解します。進捗状況、実行する必要のあるアクション、コメントなど、この問題に関連する作業用に作

成されたタスクを確認します。プルリクエスト で、オープンラベルの横にあるプルリクエストへのリンクを選択してコードを確認します。

> **Tip**
>
> タスクに対して生成されたプルリクエストは、タスクビューにリンクされたプルリクエストとしてのみ表示されます。これらは、問題のリンクされたプルリクエストとして表示されません。

3. プルリクエストで、コードの変更を確認します。詳細については、「プルリクエストの確認」を参照してください。Amazon Q で推奨コードを変更したい場合は、プルリクエストにコメントを残します。最良の結果を得るには、Amazon Q のコメントを残すときに特に注意してください。

   例えば、 用に作成されたプルリクエストを確認するときに**`Create another mysfit named Quokkapus`**、説明にタイプミスがあることに気付くかもしれません。Amazon Q に「ニーズ」と「a」の間にスペースを追加することで、「説明を変更してタイプミス」を修正「ニーズ」とコメントを残すことができます。」 または、Amazon Q に説明を更新し、変更した説明全体を記載して組み込むように指示するコメントを残すこともできます。

   新しい mysfit のイメージをウェブサイトにアップロードした場合は、Amazon Q にコメントを残して mysfit を更新し、新しい mysfit に使用するイメージとサムネイルへのポインターを付けることができます。

   > **Note**
   >
   > Amazon Q は個々のコメントには応答しません。Amazon Q は、プルリクエストのコメントに残されたフィードバックを組み込むのは、問題の作成時に承認するすべてのステップの後に停止するデフォルトのオプションを選択した場合のみです。

4. (オプション) 自分や他のプロジェクトユーザーがコードへの変更に必要なコメントをすべて残したら、リビジョンの作成を選択して、Amazon Q がコメントにリクエストした変更を組み込んだプルリクエストのリビジョンを作成します。リビジョン作成の進行状況は、Amazon Q によって概要 で報告され、変更 では報告されません。ブラウザを更新して、リビジョンの作成時に Amazon Q からの最新の更新を確認してください。

> **Note**
>
> プルリクエストのリビジョンを作成できるのは、問題を作成したユーザーのみです。プルリクエストのリビジョンは 1 つだけリクエストできます。コメントに関するすべての問題に対処していること、およびコメントの内容に満足していることを確認してから、リビジョンの作成を選択します。

5. このサンプルプロジェクトのプルリクエストごとにワークフローが実行されます。プルリクエストをマージする前に、ワークフローが正常に実行されたことを確認してください。マージする前にコードをテストするためのワークフローと環境を追加で作成することもできます。詳細については、「ワークフローの開始方法」を参照してください。
6. プルリクエストの最新リビジョンに問題がなければ、 のマージを選択します。

リソースをクリーンアップする

このチュートリアルを完了したら、次のアクションを実行して、不要になったこのチュートリアルで作成したリソースをクリーンアップすることを検討してください。

- 今後処理されなくなった問題から Amazon Q の割り当てを解除します。Amazon Q が問題に対する作業を完了したか、ソリューションを見つけられなかった場合は、生成 AI 機能の最大クォータに達しないように、Amazon Q の割り当てを解除してください。詳細については、「生成 AI 機能の管理」および「 の料金」を参照してください。
- 作業が完了した問題があれば、完了 に移動します。
- プロジェクトが不要になった場合は、プロジェクトを削除します。

チュートリアル: 構成可能な PDK ブループリントを使用したフルスタックアプリケーションの作成

Amazon CodeCatalyst には、プロジェクトをすばやく開始できるように、さまざまな設計図が用意されています。ブループリントで作成されたプロジェクトには、ソースリポジトリ、サンプルソースコード、CI/CD ワークフロー、ビルドレポートとテストレポート、統合された問題追跡ツールなど、必要なリソースが含まれています。ただし、プロジェクトを徐々に構築したり、ブループリントによって作成された既存のプロジェクトに機能を追加したりする場合があります。設計図を使用してこれを行うこともできます。このチュートリアルでは、基盤を設定し、すべてのプロジェクトコードを 1 つのリポジトリに保存できる 1 つのブループリントの使用を開始する方法を示します。そこから、

都合の良いときに最初の設計図の上に他の設計図を適用することで、追加のリソースとインフラストラクチャを柔軟に組み込むことができます。このビルディングブロック方式により、複数のプロジェクトにわたる特定の要件に対応できます。

このチュートリアルでは、複数の AWS Project Development Kit (AWS PDK) ブループリントをまとめて構成し、React ウェブサイト、Smithy API、およびそれを AWS にデプロイするためのサポート CDK インフラストラクチャで構成されるアプリケーションを作成する方法を示します。AWS PDK は、一般的なパターンの構成要素と、プロジェクトを管理および構築するための開発ツールを提供します。詳細については、「AWS PDK GitHub ソースリポジトリ」を参照してください。

次の PDK ブループリントは、組み合わせ可能な方法でアプリケーションを構築するために相互に使用するように設計されています。

- Monorepo - モノレポ内のプロジェクト間の相互依存関係を管理するルートレベルのプロジェクトを作成します。このプロジェクトでは、ビルドキャッシュと依存関係の視覚化も提供します。
- タイプセーフ API - Smithy または OpenAPI v3 で定義できる API を作成し、ビルド時のコード生成を管理して、タイプセーフな方法で API を実装して操作できるようにします。API Gateway への API のデプロイを管理し、自動入力検証を設定する CDK コンストラクトを提供します。
- Cloudscape React ウェブサイト - Cognito Auth と (オプションで) 作成された API と事前に統合されている Cloudscape を使用して構築された React ベースのウェブサイトを作成します。これにより、API を安全に呼び出すことができます。
- インフラストラクチャ - アプリケーションのデプロイに必要なすべての CDK 関連のインフラストラクチャを設定するプロジェクトを作成します。また、ビルドするたびに CDK コードに基づいて図を生成するように事前設定されています。
- DevOps - AWS Project Development Kit (AWS PDK) にあるコンストラクトと互換性のある DevOps ワークフローを作成します。

このチュートリアルには、デプロイされたアプリケーションを表示し、他のユーザーをそのアプリケーションを操作するよう招待し、プルリクエストがマージされたときに、接続された AWS アカウントのリソースに自動的に構築およびデプロイされるプルリクエストを使用してコードを変更する方法の手順も含まれています。

PDK ブループリントで構成されるプロジェクトを作成すると、プロジェクトは次のリソースを使用してプロジェクト内に CodeCatalyst 作成されます。

- モノレポとして設定されたソースリポジトリ。

- 静的コード分析とライセンスチェックを実行し、デフォルトブランチに変更が加えられるたびにサンプルコードを構築してデプロイするワークフロー。アーキテクチャ図は、コードを変更するたびに生成されます。
- 作業の計画と追跡に使用できる問題ボードとバックログ。
- 自動レポートを含むテストレポートスイート。

トピック

前提条件

プロジェクトを作成して更新するには、 でのセットアップと へのサインイン CodeCatalyst次のようにタスクを完了している必要があります。

- にサインインするための AWS Builder ID を用意します CodeCatalyst。
- スペースに属し、そのスペースにスペース管理者またはパワーユーザーロールを割り当てます。詳細については、スペースの作成、ユーザーにスペース許可を付与する、およびスペース管理者ロールを参照してください。
- スペースに AWS アカウントを関連付け、サインアップ時に作成した IAM ロールがある。例えば、サインアップ中に、CodeCatalystWorkflowDevelopmentRole-***spaceName*** ロールポリシーと呼ばれるロールポリシーを使用してサービスロールを作成するオプションがあります。ロールには一意の識別子`CodeCatalystWorkflowDevelopmentRole-`*spaceName*が付加された名前が付けられます。ロールとロールポリシーの詳細については、「」を参照してくださいCodeCatalystWorkflowDevelopmentRole-***spaceName*** サービスロールについて。ロールを作成する手順については、「」を参照してくださいアカウントとスペースのCodeCatalystWorkflowDevelopmentRole-***spaceName***ロールの作成。

ステップ 1: モノレポプロジェクトを作成する

PDK - Monorepo ブループリントから始めて、基盤として機能するモノレポコードベースを作成し、PDK ブループリントを追加できるようにします。

PDK - Monorepo ブループリントを使用してプロジェクトを作成するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. CodeCatalyst コンソールで、プロジェクトを作成するスペースに移動します。
3. スペースダッシュボードで、[プロジェクトの作成] を選択します。
4. 設計図 で開始 を選択します。
5. PDK - モノレポの設計図を選択し、次へ を選択します。
6. 「プロジェクトの名前」に、プロジェクトに割り当てる名前と、それに関連するリソース名を入力します。名前はスペース内で一意でなければなりません。
7. プロジェクトリソース で、次の操作を行います。

    a. プライマリプログラミング言語 で、プロジェクトコードを開発する言語を選択します。TypeScript、Java、または Python から選択できます。

    b. コード設定の選択

    c. ソースリポジトリのテキスト入力フィールドに、ソースリポジトリの名前を入力します。これにより、新しいリポジトリが作成されるか、既存のリンクされたリポジトリから選択します。既存のリポジトリは空である必要があります。詳細については、「ソースリポジトリのリンク」を参照してください。

    d. （オプション) パッケージマネージャーのドロップダウンメニューからパッケージマネージャーを選択します。これは、プライマリプログラミング言語 TypeScript として を選択した場合にのみ必要です。

8. （オプション) プロジェクトパラメータの選択に基づいて生成されるコードをプレビューするには、「プロジェクトプレビューの生成」から「コードの表示」を選択します。
9. （オプション) 設計図のカードから詳細を表示 を選択すると、設計図のアーキテクチャの概要、必要な接続とアクセス許可、設計図が作成するリソースの種類など、設計図に関する特定の詳細が表示されます。
10. モノレポプロジェクトを作成するには、プロジェクトの作成を選択します。作成されたルートレベルのプロジェクトは、モノレポ内のプロジェクト間の相互依存関係を管理し、ビルドキャッシュと依存関係の管理を提供します。

プロジェクトのブループリントの詳細については、「」を参照してください CodeCatalyst ブループリントを使用した包括的なプロジェクトの作成。

PDK - Monorepo ブループリントは、プロジェクトの基礎のみを生成します。ブループリントを使用して実用的なアプリケーションを作成するには、タイプセーフ API、Cloudscape React ウェブサイト、インフラストラクチャ、 などの他の PDK ブループリントを追加する必要があります DevOps。次のステップでは、Type Safe API をプロジェクトに適用します。

ステップ 2: プロジェクトに Type Safe API を追加する

PDK - タイプセーフ API ブループリントでは、Smithy または OpenAI v3 を使用して API を定義できます。API 定義からランタイムパッケージを生成します。これには、API を操作するためのクライアントと、API を実装するためのサーバー側のコードが含まれます。また、設計図では、API オペレーションごとに型安全性を備えた CDK コンストラクトも生成されます。設計図を既存の PDK モノレポプロジェクトに適用して、プロジェクトに API 機能を追加できます。

PDK を適用するには - タイプセーフ API ブループリント

1. モノレポプロジェクトのナビゲーションペインで、ブループリント を選択し、ブループリントを適用 を選択します。
2. PDK - 「安全な API 設計図」を選択し、「次へ」を選択します。
3. 「設計図の設定」で、設計図パラメータを設定します。
   - モデル言語 で、API モデルが定義されている言語を選択します。
   - 名前空間テキスト入力フィールドに、API の名前空間を入力します。
   - API 名テキスト入力フィールドに、API の名前を入力します。
   - CDK 言語 で、任意の言語を選択して、API をデプロイする CDK インフラストラクチャを記述します。
   - ハンドラー言語 (複数可) ドロップダウンメニューを選択し、API オペレーションのハンドラーを実装する言語を選択します。
   - ドキュメント形式 (複数可) ドロップダウンメニューを選択し、API ドキュメントの生成に使用する形式を選択します。
4. コード変更タブで、提案された変更を確認します。プルリクエストに表示される違いは、プルリクエストの作成時のプロジェクトへの変更を示しています。

5. ブループリントの適用時に提案された変更に満足したら、ブループリントの適用 を選択します。

   プルリクエストを作成したら、コメントを追加できます。コメントは、プルリクエスト、またはファイル内の個々の行、およびプルリクエスト全体に追加できます。ファイルなどのリソースへのリンクは、 @ 記号を使用して追加し、その後にファイルの名前を付けることができます。

   > **Note**
   >
   > ブループリントは、プルリクエストが承認されてマージされるまで適用されません。詳細については、「プルリクエストの確認」および「プルリクエストのマージ」を参照してください。

6. ステータス 列から、PDK の保留中のプルリクエスト - 安全 API ブループリント行を入力し、開いているプルリクエストのリンクを選択します。
7. マージ を選択し、希望するマージ戦略を選択し、マージ を選択して適用されたブループリントからの変更を組み込みます。

   プルリクエストがマージされると、モノレポプロジェクト内に新しいpackages/apis/*mypdkapi*フォルダが生成されます。このフォルダには、設定された Type Safe API のすべての API 関連のソースコードが含まれます。
8. ナビゲーションペインでブループリント を選択して、PDK のステータス - タイプセーフ API に最新のが表示されることを確認します。

ステップ 3: プロジェクトに Cloudscape React ウェブサイトを追加する

PDK - Cloudscape React ウェブサイトの設計図はウェブサイトを生成します。オプションの パラメータ (タイプセーフ APIsを関連付けて、認証済みのタイプセーフクライアントと、さまざまな API のテスト用のインタラクティブ API エクスプローラーをセットアップするようにウェブサイトを自動的に設定できます。

PDK - Cloudscape React ウェブサイトの設計図を適用するには

1. モノレポプロジェクトのナビゲーションペインで、ブループリント を選択し、ブループリントを適用 を選択します。
2. PDK - Cloudscape React ウェブサイトの設計図を選択し、次へ を選択します。
3. 「設計図の設定」で、設計図パラメータを設定します。

- ウェブサイト名テキスト入力フィールドに、ウェブサイトの名前を入力します。
- Type Safe APIs ドロップダウンメニューを選択し、ウェブサイト内で統合する API ブループリントを選択します。API を渡すと、認証されたクライアントがセットアップされ、必要な依存関係、API エクスプローラー、その他の機能が追加されます。

4. コード変更タブで、提案された変更を確認します。プルリクエストに表示される違いは、プルリクエストの作成時のプロジェクトへの変更を示しています。
5. ブループリントの適用時に提案された変更に満足したら、ブループリントの適用 を選択します。

   プルリクエストを作成したら、コメントを追加できます。コメントは、プルリクエスト、またはファイル内の個々の行、およびプルリクエスト全体に追加できます。ファイルなどのリソースへのリンクは、 @ 記号を使用して追加し、その後にファイルの名前を付けることができます。

   > **Note**
   >
   > ブループリントは、プルリクエストが承認されてマージされるまで適用されません。詳細については、「プルリクエストの確認」および「プルリクエストのマージ」を参照してください。

6. ステータス 列から、PDK - Cloudscape React ウェブサイトの設計図行のプルリクエスト保留中を選択し、開いているプルリクエストのリンクを選択します。
7. マージ を選択し、希望するマージ戦略を選択し、マージ を選択して適用されたブループリントからの変更を組み込みます。

   プルリクエストがマージされると、新しいウェブサイトのすべてのソースコードを含む新しい`packages/websites/`*`my-website-name`*フォルダがモノレポプロジェクト内に生成されます。
8. ナビゲーションペインでブループリント を選択して、PDK - Cloudscape React ウェブサイトのステータスに最新の が表示されることを確認します。

次に、PDK - インフラストラクチャの設計図を適用して、ウェブサイトを AWS クラウドにデプロイするインフラストラクチャを生成します。

ステップ 4: アプリケーションを AWS クラウドにデプロイするためのインフラストラクチャを生成する

PDK - インフラストラクチャの設計図は、ウェブサイトと API をデプロイするためのすべての CDK コードを含むパッケージをセットアップします。また、デフォルトで図の生成とプロトタイプナグパックへの準拠も提供します。

PDK - インフラストラクチャ設計図を適用するには

1. モノレポプロジェクトのナビゲーションペインで、ブループリント を選択し、ブループリントを適用 を選択します。
2. PDK - インフラストラクチャの設計図を選択し、次へ を選択します。
3. 「設計図の設定」で、設計図パラメータを設定します。

   - CDK 言語 で、インフラストラクチャを開発する言語を選択します。
   - スタック名テキスト入力フィールドに、設計図用に生成された CloudFormation スタックの名前を入力します。

     > **Note**
     >
     > ワークフローを設定する次のステップでは、このスタック名をメモしておきます DevOps 。

   - Type Safe APIs ドロップダウンメニューを選択し、ウェブサイト内で統合する API ブループリントを選択します。
   - Cloudscape React TS Websites ドロップダウンメニューを選択し、インフラストラクチャ内にデプロイするウェブサイトブループリント (PDK - Cloudscape React Website など) を選択します。
4. コード変更タブで、提案された変更を確認します。プルリクエストに表示される違いは、プルリクエストの作成時のプロジェクトへの変更を示しています。
5. ブループリントの適用時に提案された変更に満足したら、ブループリントの適用 を選択します。

   プルリクエストを作成したら、コメントを追加できます。コメントは、プルリクエスト、またはファイル内の個々の行、およびプルリクエスト全体に追加できます。ファイルなどのリソースへのリンクは、 @ 記号を使用して追加し、その後にファイルの名前を付けることができます。

> **Note**
>
> ブループリントは、プルリクエストが承認されてマージされるまで適用されません。詳細については、「プルリクエストの確認」および「プルリクエストのマージ」を参照してください。

6. ステータス 列から、PDK - インフラストラクチャ設計図行のプルリクエスト保留中 を選択し、開いているプルリクエストのリンクを選択します。
7. マージ を選択し、希望するマージ戦略を選択し、マージ を選択して適用されたブループリントからの変更を組み込みます。

   プルリクエストがマージされると、モノレポプロジェクト内に新しいpackages/infraフォルダが生成されます。これには、プロジェクトを AWS クラウドにデプロイするインフラストラクチャが含まれます。
8. ナビゲーションペインでブループリント を選択して、PDK - インフラストラクチャのステータスに最新の が表示されることを確認します。

次に、PDK - DevOpsブループリントを適用してアプリケーションをデプロイします。

ステップ 5: プロジェクトをデプロイする DevOps ワークフローを設定する

PDK - DevOps ブループリントは、設定で指定された AWS アカウントとロールを使用してプロジェクトを構築およびデプロイするために必要な DevOps ワークフローを生成します。

PDK DevOps ブループリントを適用するには

1. モノレポプロジェクトのナビゲーションペインで、ブループリント を選択し、ブループリントを適用 を選択します。
2. PDK - DevOpsブループリントを選択し、次へ を選択します。
3. 「設計図の設定」で、設計図パラメータを設定します。

   - 現在の環境でブートストラップ CDK を選択します。
   - スタック名テキスト入力フィールドに、デプロイする CloudFormation スタックの名前を入力します。これは、PDK - インフラストラクチャ設計図ステップ 4: アプリケーションを AWS クラウドにデプロイするためのインフラストラクチャを生成する用に で設定されたスタック名と一致する必要があります。

- AWS アカウント接続ドロップダウンメニューを選択し、リソースに使用する AWS アカウントを選択します。詳細については、「スペース AWS アカウント への の追加」を参照してください。
- アプリケーションのデプロイに使用するロールドロップダウンメニューを選択し、プロジェクトアプリケーションのデプロイに使用する IAM ロールを選択します。

> **Note**
>
> IAM ロールを作成するときは、 `SourceArn`をプロジェクト設定内の現在の `ProjectID` に制限します。詳細については、「CodeCatalystWorkflowDevelopmentRole-*spaceName* サービスロールについて」を参照してください。

- リージョンドロップダウンメニューを選択し、モノレポプロジェクトをデプロイするリージョンを選択します。デプロイは、必要な AWS サービスが存在するリージョンでのみ機能します。詳細については、「リージョン別の AWS のサービス」を参照してください。

4. コード変更タブで、提案された変更を確認します。プルリクエストに表示される違いは、プルリクエストの作成時のプロジェクトへの変更を示しています。
5. ブループリントの適用時に提案された変更に満足したら、ブループリントの適用 を選択します。

   プルリクエストを作成したら、コメントを追加できます。コメントは、プルリクエスト、またはファイル内の個々の行、およびプルリクエスト全体に追加できます。ファイルなどのリソースへのリンクは、 @ 記号を使用して追加し、その後にファイルの名前を付けることができます。

> **Note**
>
> ブループリントは、プルリクエストが承認されてマージされるまで適用されません。詳細については、「プルリクエストの確認」および「プルリクエストのマージ」を参照してください。

6. ステータス 列から、PDK - インフラストラクチャブループリント行のプルリクエスト保留中を選択し、開いているプルリクエストのリンクを選択します。
7. マージ を選択し、希望するマージ戦略を選択し、マージ を選択して適用されたブループリントからの変更を組み込みます。

プルリクエストがマージされると、モノレポプロジェクト内に新しい.codecatalyst/workflowsフォルダが生成されます。

8. ナビゲーションペインでブループリント を選択して、PDK のステータス - に最新の が表示されることを確認します。 DevOps

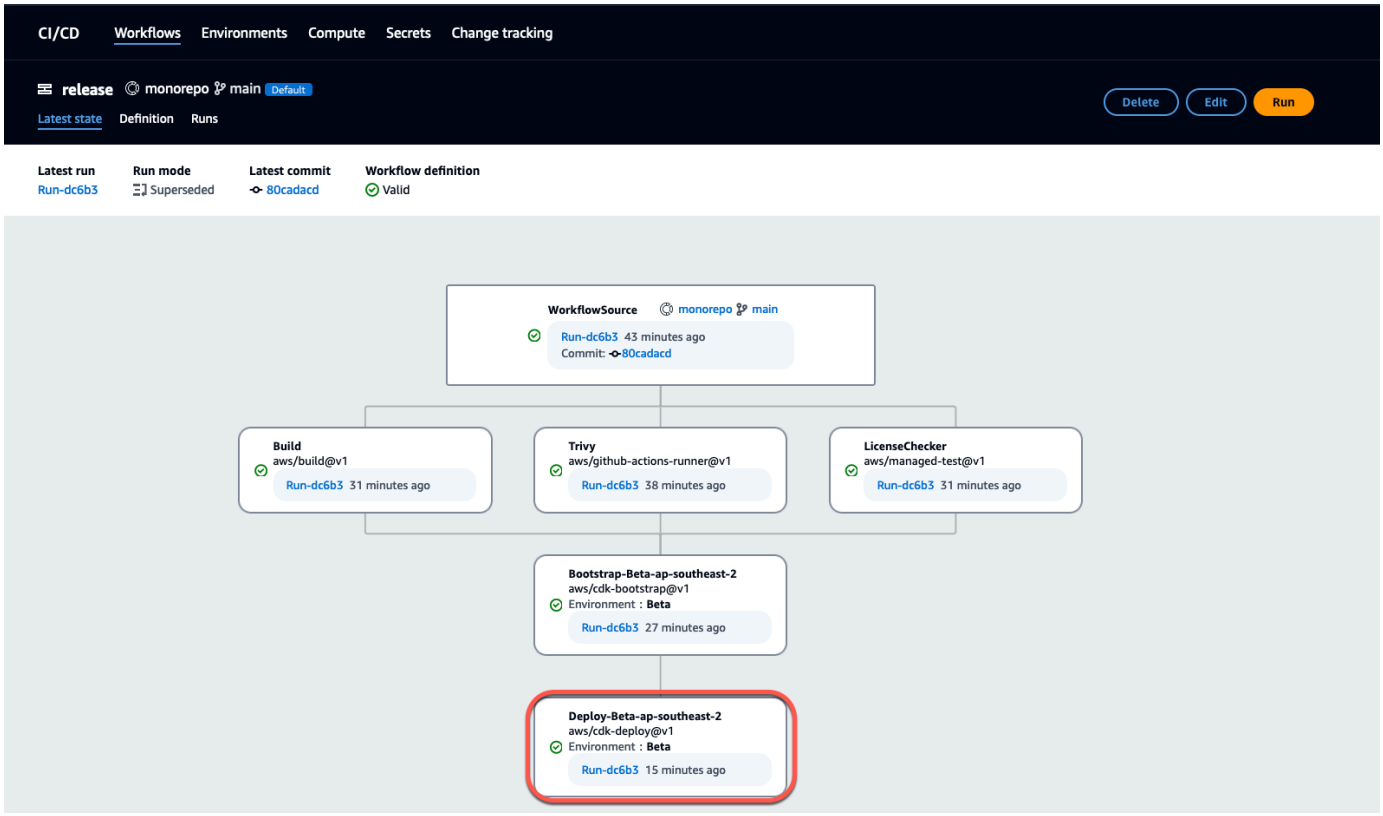
> **Note**
>
> PDK - DevOps ブループリントとそれ以降の PDK ブループリントへのすべての変更は、バックグラウンドのロックファイルが生成され、ビルドとデプロイが将来繰り返し可能になるため、この時点から大幅に遅くなります。これにより、サポートされているすべての言語のすべてのパッケージのロックファイルが生成されます。

ステップ 6: リリースワークフローを確認してウェブサイトを表示する

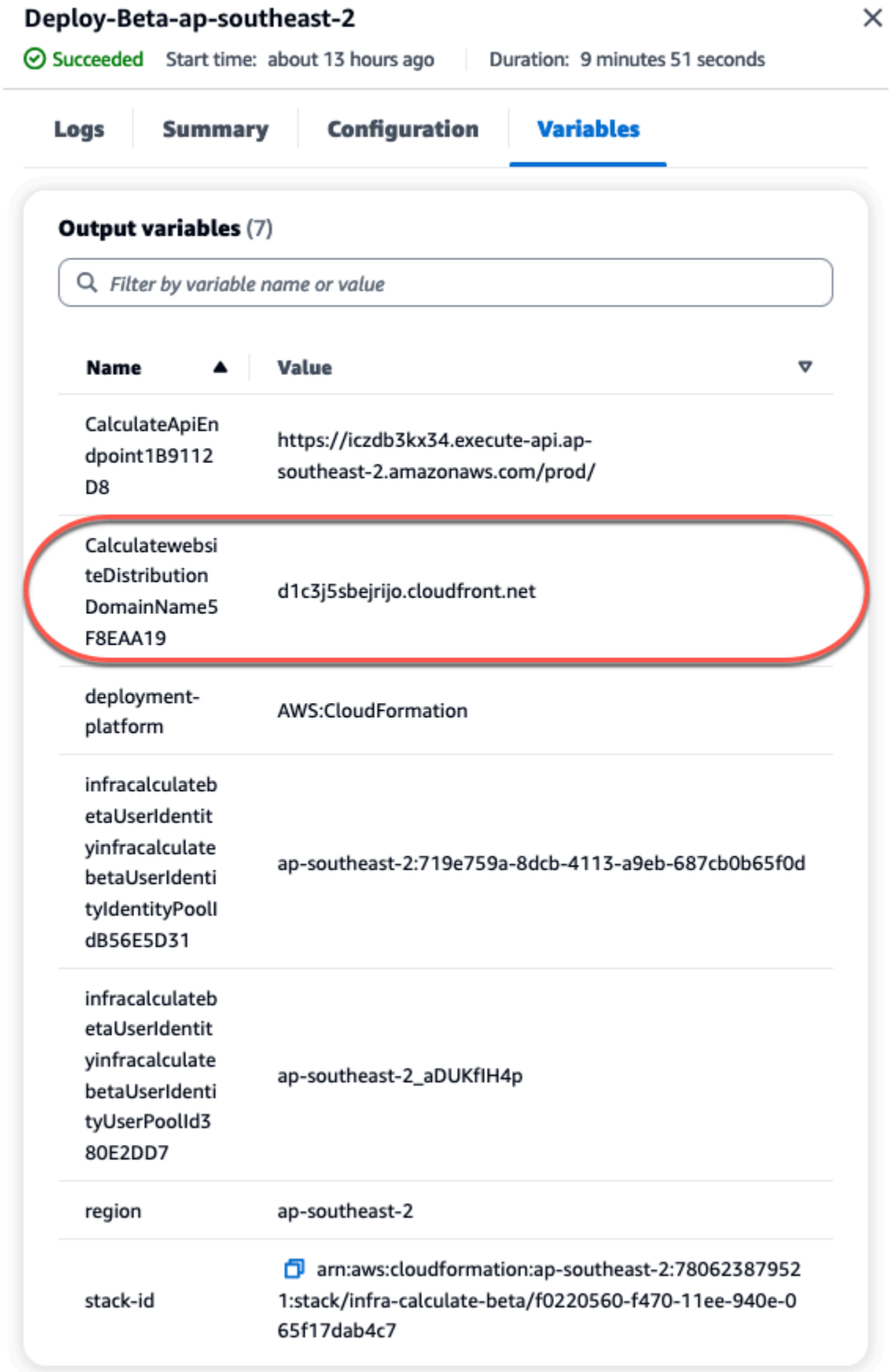
前のステップを完了したら、リリースワークフローを確認して、プロジェクトが構築されていることを確認します。

リリースワークフローを確認してウェブサイトを表示するには

1. モノレポプロジェクトのナビゲーションペインで、CI/CD を選択し、ワークフロー を選択します。
2. リリースワークフローでは、最新のワークフロー実行を選択して詳細を表示します。詳細については、「1 回の実行のステータスと詳細の表示」を参照してください。
3. ワークフローが正常に実行されたら、ワークフローの最後のアクション (Deploy-B eta-ap-souteast-2 など) を選択し、Variables を選択します。



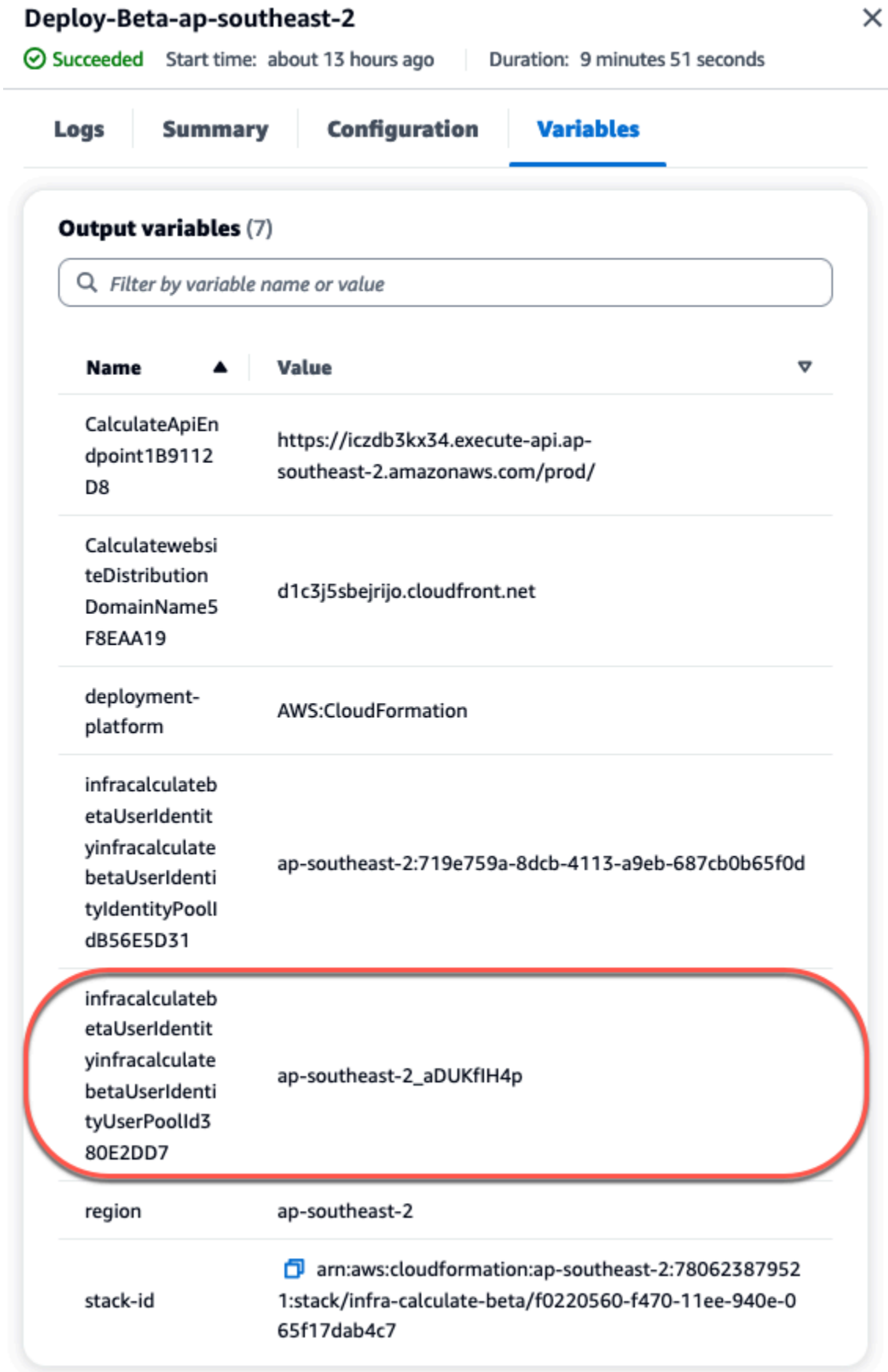
4. Variables テーブルにあるリンク (***MyPDKApi*** websiteDistributionDomainNameXXXXX など) をコピーして新しいブラウザウィンドウに貼り付けて、デプロイされたウェブサイトを表示します。


Deploy-Beta-ap-southeast-2
Succeeded
Start time: about 13 hours ago
Duration: 9 minutes 51 seconds
Logs
Summary
Configuration
Variables
Output variables (7)
Filter by variable name or value
Name
Value
CalculateApiEndpoint1B9112D8
https://iczdb3kx34.execute-api.ap-southeast-2.amazonaws.com/prod/
CalculatewebsiteDistributionDomainName5F8EAA19
d1c3j5sbejrijo.cloudfront.net
deployment-platform
AWS:CloudFormation
infracalculatebetaUserIdentityinfracalculatebetaUserIdentityIdentityPooldB56E5D31
ap-southeast-2:719e759a-8dcb-4113-a9eb-687cb0b65f0d
infracalculatebetaUserIdentityinfracalculatebetaUserIdentityUserPoolId380E2DD7
ap-southeast-2_aDUKfIH4p
region
ap-southeast-2
stack-id
arn:aws:cloudformation:ap-southeast-2:780623879521:stack/infra-calculate-beta/f0220560-f470-11ee-940e-065f17dab4c7

ウェブサイトにログインするには、Amazon Cognito アカウントが必要です。デフォルトでは、ユーザープールは自己登録を許可するように設定されていません。

a. AWS Cognito コンソール に移動します。

b. ユーザープールテーブルから、PDK - DevOpsブループリントによって作成されたユーザープールと一致するユーザープール名を選択します。これは、変数テーブルにあります (例えば、XXXXX #betaUserIdentityIdentityPoolId#####betaUserIdentityinfra。詳細については、「ユーザープールの開始方法」を参照してください。


Deploy-Beta-ap-southeast-2
Succeeded
Start time: about 13 hours ago
Duration: 9 minutes 51 seconds
Logs
Summary
Configuration
Variables
Output variables (7)
Filter by variable name or value
Name
Value
CalculateApiEndpoint1B9112D8
https://iczdb3kx34.execute-api.ap-southeast-2.amazonaws.com/prod/
CalculatewebsiteDistributionDomainName5F8EAA19
d1c3j5sbejrijo.cloudfront.net
deployment-platform
AWS:CloudFormation
infracalculatebetaUserIdentityinfracalculatebetaUserIdentityIdentityPoolIdB56E5D31
ap-southeast-2:719e759a-8dcb-4113-a9eb-687cb0b65f0d
infracalculatebetaUserIdentityinfracalculatebetaUserIdentityUserPoolId380E2DD7
ap-southeast-2_aDUKfIH4p
region
ap-southeast-2
stack-id
arn:aws:cloudformation:ap-southeast-2:780623879521:stack/infra-calculate-beta/f0220560-f470-11ee-940e-065f17dab4c7

c. [ユーザーの作成] を選択します。

d. ユーザー情報パラメータを設定します。

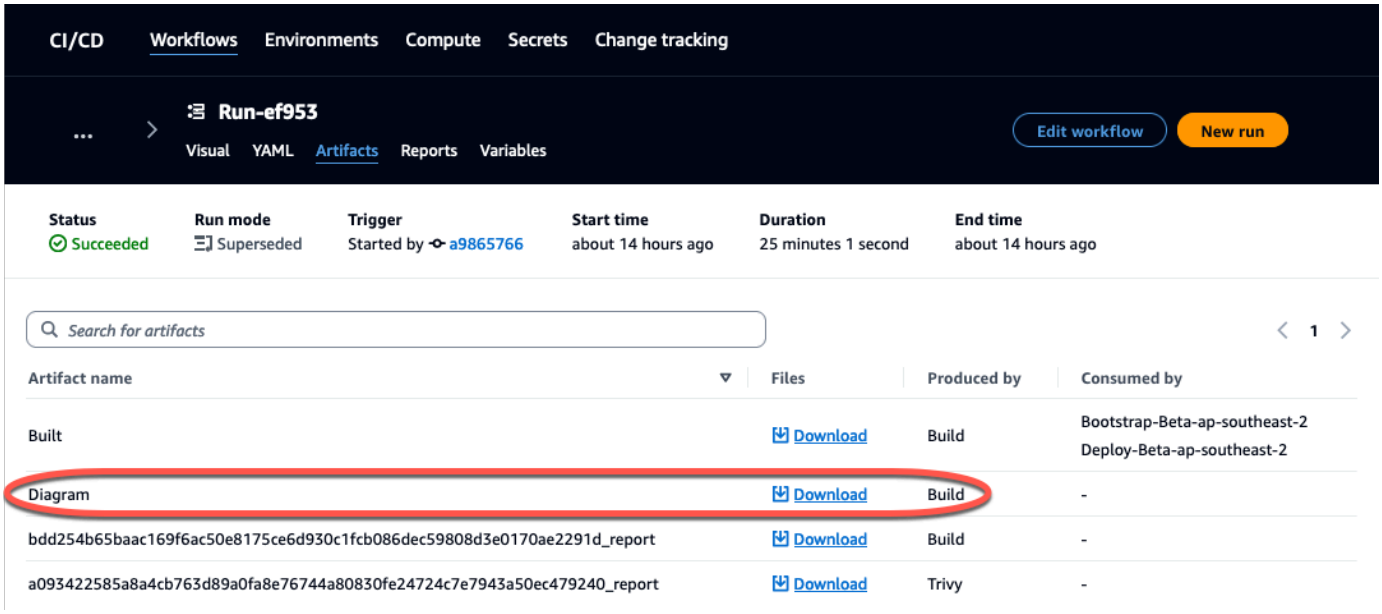
- 「招待メッセージ」で、「招待メールを送信する」を選択します。
- ユーザー名テキスト入力フィールドにユーザー名を入力します。
- E メールアドレスのテキスト入力フィールドにユーザー名を入力します。
- 一時パスワード で、パスワードの生成 を選択します。

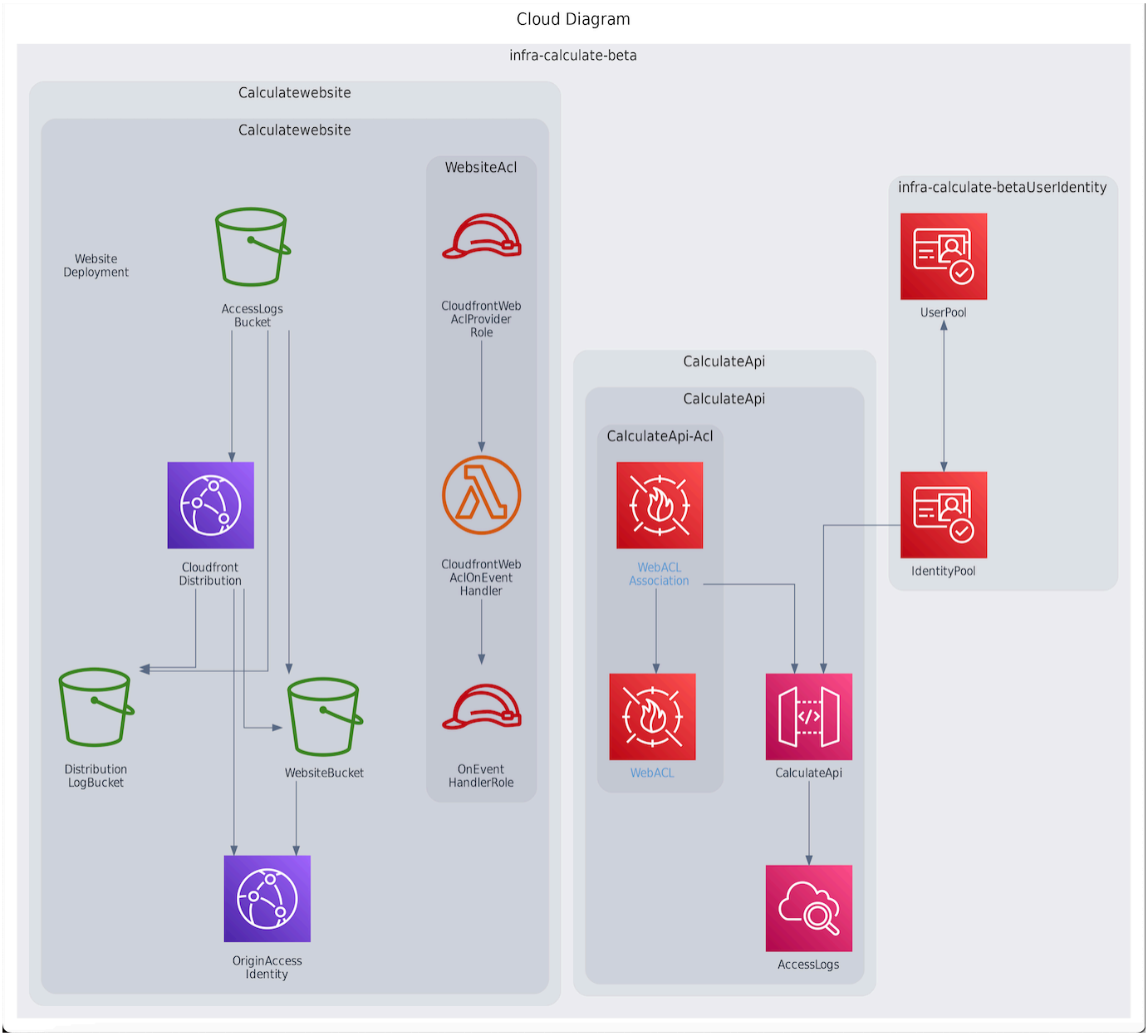
e. [ユーザーの作成] を選択します。

f. ユーザー情報パラメータに入力した E メールアカウントに移動し、一時パスワードを含む E メールを開きます。パスワードを書き留めておきます。

g. デプロイされたウェブサイトに戻り、作成したユーザー名と受け取った一時パスワードを入力し、「サインイン」を選択します。

5. (オプション) ワークフローの実行が正常に完了したら、生成された図を表示することもできます。の Artifacts タブを選択し CodeCatalyst、Download for the Diagram row を選択し、ダウンロードしたファイルを開きます (複数可)。





PDK プロジェクトのコラボレーションと繰り返し

プロジェクトを設定したら、ソースコードに変更を加えることができます。他のスペースメンバーを招待してプロジェクトに取り組むこともできます。PDK ブループリントを使用すると、各ブループリントの設定を完全に制御しながら、必要なときに必要なものだけを追加して、アプリケーションを反復的に構築できます。

トピック

- ステップ 1: メンバーをプロジェクトに招待する

- ステップ 2: 共同作業や仕事の追跡に役立つ課題を作成します。
- ステップ 3: ソースリポジトリを表示する
- ステップ 4: 開発環境を作成してコードを変更する
- ステップ 5: コードの変更をプッシュしてマージする

ステップ 1: メンバーをプロジェクトに招待する

コンソールを使用してユーザーをプロジェクトに招待できます。スペースのメンバーを招待したり、スペース外から名前を追加したりできます。

ユーザーをプロジェクトに招待するには、プロジェクト管理者またはスペース管理者ロールでサインインする必要があります。

スペース管理者ロールを持つユーザーは、すでにスペース内のすべてのプロジェクトに暗黙的にアクセスできるため、プロジェクトに招待する必要はありません。

ユーザーを (スペース管理者ロールを割り当てずに) プロジェクトに招待すると、そのユーザーは [プロジェクト] の [プロジェクトメンバー] テーブルと [スペース] の [プロジェクトメンバー] テーブルに表示されます。

プロジェクト設定タブからメンバーをプロジェクトに招待するには

1. プロジェクトに移動します。

   > **Tip**
   >
   > 上部のナビゲーションバーで表示するプロジェクトを選択できます。

2. ナビゲーションペインで [プロジェクト設定] を選択します。
3. [メンバー] タブを選択します。
4. [プロジェクトメンバー] で [新しいメンバーを招待] を選択します。
5. 新しいメンバーのメールアドレスを入力し、そのメンバーの役割を選択して、[招待] を選択します。ロールの詳細については、「ユーザーロールによるアクセス許可の付与」をご参照ください。

プロジェクト概要ページからメンバーをプロジェクトに招待するには

1. プロジェクトに移動します。

> **Tip**
>
> 上部のナビゲーションバーで表示するプロジェクトを選択できます。

2. 「メンバー +」ボタンを選択します。
3. 新しいメンバーのメールアドレスを入力し、そのメンバーの役割を選択して、[招待] を選択します。ロールの詳細については、「ユーザーロールによるアクセス許可の付与」をご参照ください。

ステップ 2: 共同作業や仕事の追跡に役立つ課題を作成します。

CodeCatalyst プロジェクトに関係する機能、タスク、バグ、その他の課題の追跡に役立ちます。課題を作成して、必要な作業やアイデアを追跡できます。デフォルトでは、課題を作成するとバックログに追加されます。課題をボードに移動して進行中の作業を追跡できます。特定のプロジェクトメンバーに課題を割り当てることもできます。このステップでは、課題を作成して PDK プロジェクトに変更を加えます。

課題を作成するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. 課題を作成したい monorepo プロジェクトに移動します。
3. プロジェクトのホームページで、「課題を作成」を選択します。または、ナビゲーションペインで [Issues] を選択します。
4. [課題の作成] を選択します。

> **Note**
>
> グリッドビューを使用すると、課題をインラインで追加することもできます。

5. 課題のタイトルを入力します。
6. (オプション) 説明を入力します。この問題については、次の説明を入力しますa change in the src/mysfit_data.json file.。Markdown を使用して書式を追加できます。
7. (オプション) 課題のステータス、優先度、見積もりを選択します。
8. (オプション) 既存のラベルを追加するか、新しいラベルを作成して [+ ラベルを追加] を選択して追加します。

a. 既存のラベルを追加するには、リストからラベルを選択します。フィールドに検索語を入力して、プロジェクト内のその用語を含むすべてのラベルを検索できます。

b. 新しいラベルを作成して追加するには、作成するラベルの名前を検索フィールドに入力し、Enter キーを押します。

9. (オプション)「+ 担当者を追加」を選択して担当者を追加します。+ Add me を選択すると、簡単に自分を担当者として追加できます。

> **Tip**
>
> Amazon Q に課題を割り当てて、Amazon Q に問題の解決を試してもらうこともできます。詳細については、「チュートリアル: CodeCatalyst 生成 AI 機能を使用して開発作業を高速化する」を参照してください。
> この機能を使用するには、そのスペースでジェネレーティブ AI 機能を有効にする必要があります。詳細については、「ジェネレーティブ AI 機能の管理」を参照してください。

10. (オプション) 既存のカスタムフィールドを追加するか、新しいカスタムフィールドを作成します。課題には複数のカスタムフィールドを設定できます。

a. 既存のカスタムフィールドを追加するには、リストからカスタムフィールドを選択します。フィールドに検索語を入力して、プロジェクト内のその用語を含むすべてのカスタムフィールドを検索できます。

b. 新しいカスタムフィールドを作成して追加するには、作成するカスタムフィールドの名前を検索フィールドに入力し、Enter キーを押します。次に、作成するカスタムフィールドの種類を選択し、値を設定します。

11. [課題を作成] を選択します。右下隅に通知が表示されます。課題が正常に作成されると、課題が正常に作成されたことを示す確認メッセージが表示されます。問題が正常に作成されなかった場合は、失敗の理由を示すエラーメッセージが表示されます。その後、[再試行] を選択して課題を編集して作成を再試行するか、[破棄] を選択して課題を破棄できます。どちらのオプションも通知を却下します。

> **Note**
>
> プルリクエストを作成したときに、そのプルリクエストを課題にリンクすることはできません。ただし、作成後に編集してプルリクエストへのリンクを追加することはできます。

詳細については、「で問題のある作業を追跡して整理する CodeCatalyst」を参照してください。

ステップ 3: ソースリポジトリを表示する

Amazon CodeCatalyst のプロジェクトに関連付けられているソースリポジトリを表示できます。のソースリポジトリの場合 CodeCatalyst、リポジトリの概要ページには、次のようなリポジトリ内の情報とアクティビティの概要が簡単に表示されます。

- リポジトリの説明 (ある場合)
- リポジトリ内のブランチ数。
- リポジトリのオープンプルリクエストの数。
- リポジトリの関連ワークフローの数。
- デフォルトブランチ、または選択したブランチ内のファイルとフォルダー
- 表示されたブランチへの最後のコミットのタイトル、作成者、日付
- マークダウンでレンダリングされた README.md ファイルの内容 (README.md ファイルが含まれている場合)

このページには、リポジトリのコミット、ブランチ、プルリクエストへのリンクのほか、個々のファイルをすばやく開き、表示、編集する方法も記載されています。

> **Note**
>
> リンクされたリポジトリに関するこの情報は、コンソールでは表示できません。CodeCatalyst リンクされたリポジトリに関する情報を表示するには、リポジトリのリストからリンクを選択し、そのリポジトリをホストするサービスでそのリポジトリを開きます。

プロジェクトのソースリポジトリに移動するには

1. プロジェクトに移動し、次のいずれかを実行します。
   - プロジェクトの概要ページで、一覧から目的のリポジトリを選択し、[リポジトリを表示] を選択します。
   - ナビゲーションペインで [コード] を選択してから、[ソースリポジトリ] を選択します。「ソースリポジトリ」で、リストからリポジトリの名前を選択します。フィルターバーにリポジトリ名の一部を入力することで、リポジトリのリストをフィルタリングできます。

2. リポジトリのホームページには、リポジトリの内容と、プルリクエストの数やワークフローなどの関連リソースに関する情報が表示されます。デフォルトでは、デフォルトブランチのコンテンツが表示されます。ドロップダウンリストから別のブランチを選択することでビューを変更できます。

> **Tip**
>
> また、プロジェクト概要ページから「プロジェクトコードを表示」を選択すれば、プロジェクトのリポジトリにすばやく移動できます。

ステップ 4: 開発環境を作成してコードを変更する

このステップでは、開発環境を作成してコードを変更し、それをメインブランチにマージします。このチュートリアルでは簡単な AWS PDK プロジェクトについて説明しますが、AWS GitHub PDK リポジトリで提供されているより複雑な例に従うこともできます。

新しいブランチで開発環境を作成するには

1. monorepo プロジェクトのナビゲーションペインで、次のいずれかを実行します。
   - 「概要」を選択し、「マイ開発環境」セクションに移動します。
   - 「コード」を選択し、「開発環境」を選択します。
   - [コード] を選択し、[ソースリポジトリ] を選択してから、開発環境を作成する monorepo リポジトリを選択します。
2. サポートされている IDE をドロップダウンメニューから選択します。詳細については、「開発環境でサポートされる統合開発環境」を参照してください。
3. [リポジトリのクローン] を選択します。
4. クローンするリポジトリを選択し、[新しいブランチで作業する] を選択し、[ブランチ名] フィールドにブランチ名を入力し、[ブランチの作成元] ドロップダウンメニューから新しいブランチを作成するブランチを選択します。

> **Note**
>
> 「ソースリポジトリ」ページまたは特定のソースリポジトリから開発環境を作成する場合、リポジトリを選択する必要はありません。開発環境は、ソースリポジトリページで選択したソースリポジトリから作成されます。

5. (オプション)「エイリアス-オプション」に、開発環境のエイリアスを入力します。
6. (オプション) 開発環境の構成編集ボタンを選択して、開発環境のコンピューティング、ストレージ、またはタイムアウト構成を編集します。
7. (オプション) Amazon Virtual Private Cloud (Amazon VPC)-オプション) で、ドロップダウンメニューから開発環境に関連付ける VPC 接続を選択します。

   スペースにデフォルト VPC が設定されている場合、開発環境はその VPC に接続して実行されます。別の VPC 接続を関連付けることでこれを無効にできます。また、VPC 接続の開発環境は AWS Toolkit をサポートしていないことにも注意してください。

   > **Note**
   >
   > VPC 接続を使用して開発環境を作成すると、VPC 内に新しいネットワークインターフェイスが作成されます。 CodeCatalyst 関連する VPC ロールを使用してこのインターフェースと対話します。また、IPv4 CIDR ブロックが IP アドレス範囲に設定されていないことも確認してください。`172.16.0.0/12`

8. [作成] を選択します。開発環境の作成中は、開発環境のステータス列に [開始中] と表示され、開発環境が作成されると、ステータス列に [実行中] と表示されます。

Dev Environment が実行されたら、 CodeCatalyst プルリクエストを使用してコードに変更を加えることで、生成されたサンプルアプリケーションを操作できます。プルリクエストは、プルリクエストがマージされると自動的にビルドされ、接続された AWS アカウントのリソースにデプロイされます。monorepo は devfile を販売するので、必要なグローバル依存関係とランタイムはすべて自動的に作成されます。

プロジェクト内のコードを変更するには

1. Dev Environment の作業ターミナルで monorepo プロジェクトに移動し、以下のコマンドを実行してプロジェクトの依存関係をインストールします。

```
npx projen install
```

2. API packages/apis/*mypdkapi*/model/src/main/smithy/operations/say-hello.smithy オペレーションの例を定義しているに移動します。このチュートリアルでは、2 Calculate つの数値を加算する簡単なオペレーションを作成します。この演算を定義するコードに、入力と出力を含めて変更を加えます。

   例:

```
$version: "2"
namespace com.aws

@http(method: "POST", uri: "/calculate")
@handler(language: "typescript")
operation Calculate {
    input := {
        @required
        numberA: Integer
        @required
        numberB: Integer
    }
    output := {
        @required
        result: Integer
    }
}
```

   @handlerこの特性は、このオペレーションを AWS Lambda ハンドラーで記述されたものとして実装することを Type Safe API に伝えます。 TypeScriptType Safe API は、このオペレーションのスタブを生成して実装できるようにします。 TypeScript@requiredトレイトが追加されます。つまり、デプロイされる API ゲートウェイによって実行時にそのトレイトが適用されます。詳細については、Smithy のドキュメントを参照してください。

3. /say-hello.smithyファイル名は、コードの変更と一致する名前 (例:) に変更します。calculate.smithy

4. に移動しpackages/apis/*mypdkapi*/model/src/main/smithy/main.smithy、コードを変更して操作を連携させます。Calculateで定義されているオペレーションは、/calculate.smithyoperationsこのファイルのフィールドにリストすることで公開できます。

例:

```
$version: "2"
namespace com.aws

use aws.protocols#restJson1

/// A sample smithy api
@restJson1
service MyPDKApi {
    version: "1.0"
    operations: [Calculate]
    errors: [
      BadRequestError
      NotAuthorizedError
      InternalFailureError
    ]
}
```

5. 以下のコマンドを実行して変更を作成します。

```
npx projen build
```

> **Note**
>
> オプションで --parallel X flag を渡すこともできます。フラグを渡すと、Xビルドが複数のコアに分散されます。

@handlerトレイトが追加されたので、ビルドが完了すると以下のファイルが生成されます。

- /packages/apis/*mypdkapi*/handlers/typescript/src/*calculate.ts*
- /packages/apis/*mypdkapi*/handlers/typescript/test/*calculate.test.ts*

6. に移動しpackages/apis/*mypdkapi*/handlers/typescript/src/*calculate.ts*、コードを変更します。このファイルは API に対して呼び出されるサーバーハンドラーです。

```
import {
  calculateHandler,
  CalculateChainedHandlerFunction,
```

```
  INTERCEPTORS,
  Response,
  LoggingInterceptor,
} from 'mypdkapi-typescript-runtime';

/**
 * Type-safe handler for the Calculate operation
 */
export const calculate: CalculateChainedHandlerFunction = async (request) => {
  LoggingInterceptor.getLogger(request).info('Start Calculate Operation');

  const { input } = request;

  return Response.success({
    result: input.body.numberA + input.body.numberB,
  });
};

/**
 * Entry point for the AWS Lambda handler for the Calculate operation.
 * The calculateHandler method wraps the type-safe handler and manages marshalling
 inputs and outputs
 */
export const handler = calculateHandler(...INTERCEPTORS, calculate);
```

7. /packages/apis/*mypdkapi*/handlers/typescript/test/*calculate.test.ts*ファイルに移動し、コードを変更してユニットテストを更新します。

   例:

```
import {
  CalculateChainedRequestInput,
  CalculateResponseContent,
} from 'mypdkapi-typescript-runtime';
import {
  calculate,
} from '../src/calculate';

// Common request arguments
const requestArguments = {
  chain: undefined as never,
  event: {} as any,
  context: {} as any,
```

```
  interceptorContext: {
    logger: {
      info: jest.fn(),
    },
  },
} satisfies Omit<CalculateChainedRequestInput, 'input'>;

describe('Calculate', () => {

  it('should return correct sum', async () => {
    const response = await calculate({
      ...requestArguments,
      input: {
        requestParameters: {},
        body: {
          numberA: 1,
          numberB: 2
        }
      },
    });

    expect(response.statusCode).toBe(200);
    expect((response.body as CalculateResponseContent).result).toEqual(3);
  });

});
```

8. /packages/infra/main/src/constructs/apis/*mypdkapi.ts*ファイルに移動し、コードを変更して CDK Calculate インフラストラクチャーに操作用のインテグレーションを追加します。API コンストラクトには統合プロパティがあり、前に追加した実装を渡すことができます。Smithy @handler Calculate モデルの特性を操作に使用しているので、生成された CalculateFunction CDK コンストラクト (事前設定済み) をハンドラー実装の指しとして使用できます。

   例:

```
import { UserIdentity } from "@aws/pdk/identity";
import { Authorizers, Integrations } from "@aws/pdk/type-safe-api";
import { Stack } from "aws-cdk-lib";
import { Cors } from "aws-cdk-lib/aws-apigateway";
import {
  AccountPrincipal,
  AnyPrincipal,
```

```
  Effect,
  PolicyDocument,
  PolicyStatement,
} from "aws-cdk-lib/aws-iam";
import { Construct } from "constructs";
import { Api, CalculateFunction } from "calculateapi-typescript-infra";

/**
 * Api construct props.
 */
export interface CalculateApiProps {
  /**
   * Instance of the UserIdentity.
   */
  readonly userIdentity: UserIdentity;
}

/**
 * Infrastructure construct to deploy a Type Safe API.
 */
export class CalculateApi extends Construct {
  /**
   * API instance
   */
  public readonly api: Api;

  constructor(scope: Construct, id: string, props?: CalculateApiProps) {
    super(scope, id);

    this.api = new Api(this, id, {
      defaultAuthorizer: Authorizers.iam(),
      corsOptions: {
        allowOrigins: Cors.ALL_ORIGINS,
        allowMethods: Cors.ALL_METHODS,
      },
      integrations: {
        calculate: {
          integration: Integrations.lambda(new CalculateFunction(this,
 "CalculateFunction"))
        }
      },
      policy: new PolicyDocument({
        statements: [
```

```
          // Here we grant any AWS credentials from the account that the prototype
 is deployed in to call the api.
          // Machine to machine fine-grained access can be defined here using more
 specific principals (eg roles or
          // users) and resources (ie which api paths may be invoked by which
 principal) if required.
          // If doing so, the cognito identity pool authenticated role must still
 be granted access for cognito users to
          // still be granted access to the API.
          new PolicyStatement({
            effect: Effect.ALLOW,
            principals: [new AccountPrincipal(Stack.of(this).account)],
            actions: ["execute-api:Invoke"],
            resources: ["execute-api:/*"],
          }),
          // Open up OPTIONS to allow browsers to make unauthenticated preflight
 requests
          new PolicyStatement({
            effect: Effect.ALLOW,
            principals: [new AnyPrincipal()],
            actions: ["execute-api:Invoke"],
            resources: ["execute-api:/*/OPTIONS/*"],
          }),
        ],
      }),
    });

    // Grant authenticated users access to invoke the api
    props?.userIdentity.identityPool.authenticatedRole.addToPrincipalPolicy(
      new PolicyStatement({
        effect: Effect.ALLOW,
        actions: ["execute-api:Invoke"],
        resources: [this.api.api.arnForExecuteApi("*", "/*", "*")],
      }),
    );
  }
}
```

9. 以下のコマンドを実行して変更を作成します。

```
npx projen build
```

プロジェクトの構築が完了すると、更新された生成されたダイアグラムを表示できます。ダイアグラムはにあります/packages/infra/main/cdk.out/cdkgraph/diagram.png。この図は、作成した API に関数がどのように追加され、接続されるかを示しています。CDK コードが変更されると、この図も更新されます。

変更内容をリポジトリのメインブランチにプッシュしてマージすることでデプロイできるようになりました。

ステップ 5: コードの変更をプッシュしてマージする

コードの変更をコミットしてプッシュし、ソースリポジトリのメインブランチにマージできます。

フィーチャーブランチに変更をプッシュするには

- 以下のコマンドを実行して、変更をコミットしてフィーチャーブランチにプッシュします。

```
git add .
```

```
git commit -m "my commit message"
```

```
git push
```

変更をプッシュすると、フィーチャーブランチの新しいワークフローが実行され、 CodeCatalyst コンソールで確認できます。その後、プルリクエストを作成して、変更をソースリポジトリのメインブランチにマージできます。フィーチャーブランチをメインブランチにマージすると、リリースワークフローがトリガーされます。プルリクエストを課題にリンクすることもできます。

プルリクエストを作成して課題にリンクするには

1. monorepo プロジェクトで、以下のいずれかを実行してください。
   - ナビゲーションペインで [コード] を選択し、[プルリクエスト] を選択し、[プルリクエストの作成] を選択します。
   - リポジトリのホームページで [More] を選択し、[Create pull request (プルリクエストの作成)] を選択します。
   - プロジェクトページで [プルリクエストを作成] を選択します。

2. [ソースリポジトリ] で、指定したソースリポジトリがコミットされたコードを含むソースリポジトリであることを確認します。このオプションは、リポジトリのメインページからプルリクエストを作成しなかった場合にのみ表示されます。
3. 「宛先ブランチ」で、レビュー後にコードをマージするメインブランチを選択します。
4. 「ソースブランチ」で、コミットされたコードを含むフィーチャーブランチを選択します。
5. プルリクエストのタイトルには、レビューが必要な内容とその理由を他のユーザーが理解しやすいタイトルを入力します。
6. (オプション) プルリクエストの説明には、課題へのリンクや変更内容の説明などの情報を入力します。

> **Tip**
>
> 「説明を書く」を選択すると、 CodeCatalyst プルリクエストに含まれる変更の説明が自動的に生成されます。自動生成された説明は、プルリクエストに追加した後で変更できます。
> この機能を使用するには、そのスペースでジェネレーティブ AI 機能を有効にする必要があります。詳細については、「Amazon CodeCatalyst におけるジェネレーティブ AI 機能の管理」を参照してください。

7. [課題] で [課題をリンク] を選択し、ステップ 2: 共同作業や仕事の追跡に役立つ課題を作成します。で作成した課題を選択します。課題をリンク解除するには、リンク解除アイコンを選択します。
8. (オプション)「必須のレビュー担当者を追加」で、「必須のレビュー担当者を追加」を選択します。プロジェクトメンバーのリストから選択して追加します。プルリクエストをターゲットブランチにマージする前に、必須のレビュー担当者が変更を承認する必要があります。

> **Note**
>
> レビュー担当者を必須のレビュー担当者とオプションのレビュー担当者の両方として追加することはできません。自分をレビュー担当者として追加することはできません。

9. (オプション)「任意のレビュー担当者を追加」で、「任意のレビュー担当者を追加」を選択します。プロジェクトメンバーのリストから選択して追加します。プルリクエストをターゲットブランチにマージする前に、任意のレビュアーが変更を要件として承認する必要はありません。
10. プルリクエストはレビュアーまたは自分でレビューしてメインブランチにマージする必要があります。詳細については、「プルリクエストのマージ」を参照してください。

変更がソースリポジトリのメインブランチにマージされると、新しいワークフローが自動的にトリガーされます。

11. マージが完了したら、Issue を Done に移動できます。

    a. ナビゲーションペインで [Issues] を選択します。

    b. で作成した課題を選択しステップ 2: 共同作業や仕事の追跡に役立つ課題を作成します。、「ステータス」ドロップダウンを選択して、「完了」を選択します。

リリースワークフローでは、実行が成功するとアプリケーションがデプロイされるので、変更を確認できます。

リリースワークフローを確認して Web サイトを表示するには

1. monorepo プロジェクトのナビゲーションペインで、「CI/CD」を選択し、「ワークフロー」を選択します。
2. リリースワークフローでは、最後に実行されたワークフローを選択して詳細を表示します。詳細については、「1 回の実行のステータスと詳細の表示」を参照してください。
3. ワークフローの実行が正常に完了したら、ワークフローの最後のアクション (Deploy-B eta-ap-southeast -2) を選択し、[変数] を選択します。
4. ***MyPDKAPI*** websiteDistributionDomain NameXXXxx 行のリンクをコピーして新しいブラウザーウィンドウに貼り付けて、デプロイされた Web サイトを表示します。
5. 作成したユーザー名とパスワードを入力し、[Sign in] を選択します。ステップ 6: リリースワークフローを確認してウェブサイトを表示する
6. (オプション) アプリケーションの変更をテストします。

    a. POST ドロップダウンメニューを選択します。

    b. `numberA`とに 2 つの値を入力し`number B`、[実行] を選択します。

    c. レスポンスボディで結果を確認します。

時間の経過とともに、PDK ブループリントのカタログバージョンは変わる可能性があります。プロジェクトのブループリントをカタログバージョンに変更して、最新の変更に遅れないようにすることができます。プロジェクトのブループリントバージョンを変更する前に、コードの変更と影響を受ける環境を確認できます。詳細については、「プロジェクトのブループリントバージョンの変更」を参照してください。

のスペースでリソースを整理する CodeCatalyst

自分、会社、部門、またはグループを表すスペースを作成し、開発チームがプロジェクトを管理できる場所を提供します。Amazon で作成するプロジェクト、メンバー、および関連するクラウドリソースを追加するスペースを作成する必要があります CodeCatalyst。

> **Note**
>
> スペース名は 全体で一意である必要があります CodeCatalyst。削除されたスペースの名前は再利用できません。

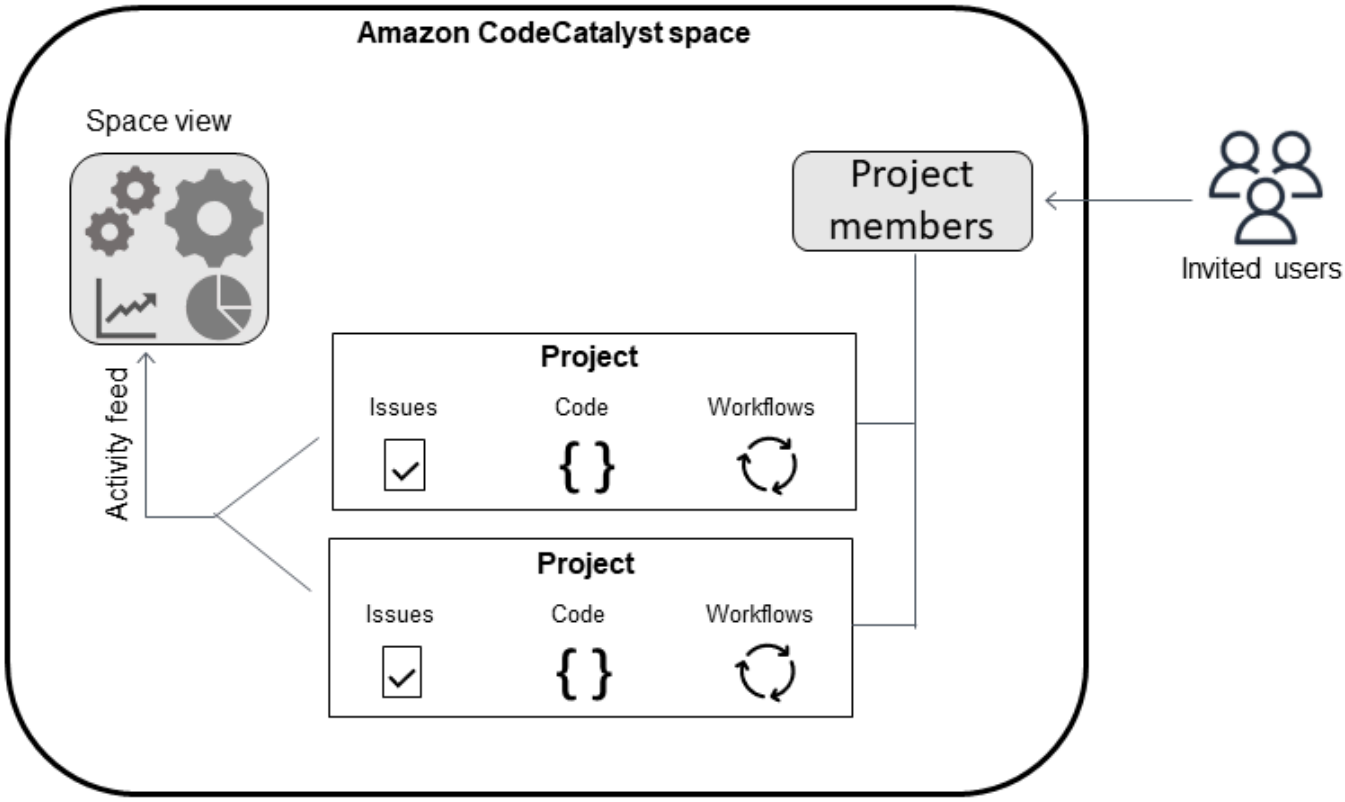
スペースを作成すると、スペース管理者ロールが自動的に割り当てられます。このロールは、スペース内の他のユーザーに追加できます。

Space 管理者ロールを使用すると、次のようにスペースを管理できます。

- スペースに他のスペース管理者を追加する
- メンバーのロールとアクセス許可を変更する
- スペースの編集または削除
- プロジェクトを作成し、メンバーをプロジェクトに招待する
- スペース内のすべてのプロジェクトのリストを表示する
- スペース内のすべてのプロジェクトのアクティビティフィードを表示する

スペースを作成すると、スペース管理者ロールと、スペースの作成の一環として作成したプロジェクトのプロジェクト管理者ロールの 2 つのロールを使用して、自動的にスペースに追加されます。プロジェクトへの招待を受け入れると、追加のユーザーがメンバーとして自動的にスペースに追加されます。スペース内のこのメンバーシップは、スペース内のアクセス許可を付与しません。ユーザーがスペースで実行できる操作は、特定のプロジェクトでユーザーが持つロールによって決まります。

ロールの詳細については、「ユーザーロールによるアクセス許可の付与」をご参照ください。



追加されたアカウントに関する追加の考慮事項は次のとおりです。

- スペース AWS アカウント のアカウント接続の への one-to-one マッピングがあります。1 つの を複数の異なるスペースに追加 AWS アカウント できます。デプロイする AWS アカウントは一意である必要はなく、複数のスペースで使用できます。
- AWS アカウント CodeCatalyst スペースに追加された は、そのスペース内の任意のプロジェクトで使用できます。
- 各環境は複数の をサポートできますが AWS アカウント、 アクションでは環境ごとに 1 つのアカウントしか使用できません。
- 請求はスペースレベルで設定されます。請求用に複数のアカウントを設定できますが、1 つの CodeCatalyst スペースでアクティブにできるアカウントは 1 つだけです。のスペースの請求アカウントとして使用できるの AWS アカウント は 1 つだけです CodeCatalyst。アカウントが既にスペースに使用されている場合は、追加のスペースに別の請求アカウントを使用する必要があります。
- 接続を作成したら、ワークフローが CodeCatalyst 環境でそれらの AWS IAM ロールにアクセスする必要がある場合は、接続に IAM ロールを追加する必要があります。環境の使用方法の詳細につ

いては、「」を参照してください環境を使用して AWS アカウント および VPCsにデプロイする CodeCatalyst 。

トピック

スペースの作成

Builder ID CodeCatalyst を使用して Amazon AWS に初めてサインアップするときは、スペースを作成する必要があります。詳細については、「のセットアップと へのサインイン CodeCatalyst」を参照してください。ビジネスニーズに合わせて追加のスペースを作成することもできます。

> **Note**
>
> スペース名は 全体で一意である必要があります CodeCatalyst。削除されたスペースの名前は再利用できません。

このガイドの情報は、 Builder ID ユーザー CodeCatalyst をサポートするスペースを AWS に作成するために提供されています。ID フェデレーションをサポートするスペースを設定および管理するためのステップは、 CodeCatalyst 管理者ガイドに記載されています。ID フェデレーション用に設定されたスペースを操作するには、「Amazon CodeCatalyst 管理者ガイド」の CodeCatalyst 「スペースのセットアップと管理」を参照してください。

AWS Builder ID ユーザーをサポートする追加のスペースを作成するには、スペース管理者ロールを割り当てる必要があります。

> **Note**
>
> 追加のスペースを作成する場合、プロジェクトを作成するように求められません。スペースにプロジェクトを作成する方法については、「」を参照してください「プロジェクトの作成」。

別のスペースを作成するには

1. で AWS Management Console、 CodeCatalyst スペースに関連付けるの AWS アカウント と同じ でサインインしていることを確認します。
2. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
3. 自分のスペースに移動します。

   > **Tip**
   >
   > 複数のスペースに属している場合は、上部のナビゲーションバーでスペースを選択します。

4. スペースの作成 を選択します。
5. 「スペースの作成」ページの「スペース名」に、スペースの名前を入力します。これは後で変更することはできません。

   > **Note**
   >
   > スペース名は 全体で一意である必要があります CodeCatalyst。削除されたスペースの名前は再利用できません。

6. でAWS リージョン、スペースとプロジェクトデータを保存するリージョンを選択します。これは後で変更することはできません。
7. AWS アカウント ID に、スペースに接続するアカウントの 12 桁の ID を入力します。

   [AWS アカウント確認トークン] に、生成されたトークン ID をコピーします。トークンは自動的にコピーされますが、 AWS 接続リクエストの承認中に保存することもできます。
8. で検証を選択します AWS。

9. で「Amazon CodeCatalyst スペースの検証」ページが開きます AWS Management Console。これは Amazon CodeCatalyst Spaces ページです。ページにアクセスするには、サインインが必要な場合があります。

   で AWS Management Console、スペースを作成する AWS リージョン 場所と同じ を選択してください。

   ページに直接アクセスするには、https://console.aws.amazon.com/codecatalyst/home/. の で Amazon CodeCatalyst Spaces にサインイン AWS Management Console します。

   検証トークンは検証トークン に自動的に入力されます。成功バナーには、トークンが有効なトークンであることを示すメッセージが表示されます。

10. [スペースを確認] を選択します。

    アカウントがスペースに追加されたことを示すアカウント検証の成功メッセージが表示されます。

11. Amazon CodeCatalyst スペースの検証ページにとどまります。次のリンクを選択します。このスペースに IAM ロールを追加するには、スペースの詳細を表示します。

    でCodeCatalyst スペースの詳細ページが開きます AWS Management Console。これは Amazon CodeCatalyst Spaces ページです。ページにアクセスするには、ログインが必要な場合があります。

12. で使用できる IAM ロール で CodeCatalyst、IAM ロールの追加 を選択します。

    ページで使用できる IAM ロールの追加 CodeCatalyst が表示されます。

13. IAM で CodeCatalyst 開発管理者ロールの作成を選択します。このオプションは、開発ロールのアクセス許可ポリシーと信頼ポリシーを含むサービスロールを作成します。

    デベロッパーロールは、 CodeCatalyst ワークフローが Amazon S3、Lambda、 などの AWS リソースにアクセスできるようにする AWS IAM ロールです AWS CloudFormation。ロールには一意の識別子`CodeCatalystWorkflowDevelopmentRole-`*`spaceName`*が付加された名前が付けられます。ロールとロールポリシーの詳細については、「」を参照してくださいCodeCatalystWorkflowDevelopmentRole-***spaceName*** サービスロールについて。

14. 開発ロールの作成 を選択します。

15. 接続ページの で使用できる IAM ロールで CodeCatalyst、アカウントに追加された IAM ロールのリストでデベロッパーロールを表示します。

16. Amazon に移動 CodeCatalystを選択します。

17. の作成ページで CodeCatalyst、スペースの作成 を選択します。

スペースの編集

スペースの説明を変更して、ユーザーがスペースの内容をよりよく理解できるようにすることができます。

スペースの詳細を編集するには、スペース管理者ロールが必要です。

このガイドの情報は、 AWS Builder ID ユーザー CodeCatalyst をサポートする のスペースを編集するために提供されています。ID フェデレーションをサポートするスペースを設定および管理するためのステップの詳細については、Amazon CodeCatalyst 管理者ガイドの CodeCatalyst 「スペースのセットアップと管理」を参照してください。

スペースの説明を編集するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 自分のスペースに移動します。

> **Tip**
>
> 複数のスペースに属している場合は、上部のナビゲーションバーでスペースを選択します。

3. スペース設定 タブで、編集 を選択します。スペースの説明に変更を加え、保存 を選択します。

スペースの削除

スペースを削除して、そのスペースのすべてのリソースへのアクセスを削除できます。スペースを削除するには、スペース管理者ロールが必要です。

> **Note**
>
> スペースの削除を元に戻すことはできません。

スペースを削除すると、すべてのスペースメンバーがスペースリソースにアクセスできなくなります。スペースリソースの請求も停止し、サードパーティーのソースリポジトリによってプロンプトされるワークフローはすべて停止します。

> **Note**
>
> スペース名は 全体で一意である必要があります CodeCatalyst。削除されたスペースの名前は再利用できません。

このガイドの情報は、 AWS Builder ID ユーザー CodeCatalyst をサポートする のスペースを削除するために提供されています。ID フェデレーションをサポートするスペースを設定および管理するためのステップの詳細については、Amazon CodeCatalyst 管理者ガイドの CodeCatalyst「スペースのセットアップと管理」を参照してください。

スペースを削除するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 自分のスペースに移動します。

   > **Tip**
   >
   > 複数のスペースに属している場合は、上部のナビゲーションバーでスペースを選択します。

3. 設定 を選択し、削除 を選択します。
4. 削除**delete**を確認するには、「」と入力します。
5. [削除] を選択します。

   > **Note**
   >
   > 複数のスペースに属している場合は、スペースの概要ページにリダイレクトされます。1 つのスペースに属している場合は、スペース作成ページにリダイレクトされます。

スペース内のユーザーとリソースのアクティビティのモニタリング

最近作成されたプロジェクトとステータスの更新を表示するには、 CodeCatalyst コンソールを使用して、スペースリソースの更新を示すアクティビティフィードを表示できます。

アクティビティフィードでは、失敗したワークフロー実行や作成されたプロジェクトなどのメトリクスを表示できます。

スペース内のアクティビティを表示するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. CodeCatalyst スペースに移動します。

> **Tip**
>
> 複数のスペースに属している場合は、上部のナビゲーションバーでスペースを選択します。

3. [Activity] (アクティビティ) を選択します。
4. アクティビティ の情報を表示します。
5. アクティビティでフィルタリングするには、右上のセレクターを選択します。
6. スペース内のすべてのアクティビティを表示するには、任意のアクティビティタイプ を選択します。

接続された AWS リソースへのアクセスを許可する AWS アカウント

Amazon CodeCatalyst スペース AWS アカウント で のリソースを使用できます。そのためには、AWS アカウント と のスペース間の接続を設定する必要があります CodeCatalyst。このような接続を作成すると、 CodeCatalyst スペース内のプロジェクトやワークフローが のリソースとやり取りできるようになります AWS アカウント。 CodeCatalyst スペース AWS アカウント で使用する各 に 1 つの接続を作成する必要があります。

接続を作成したら、 AWS IAM ロールを関連付けることを選択できます。

トピック

- スペース AWS アカウント への の追加
- アカウント接続への IAM ロールの追加
- デプロイ環境にアカウント接続と IAM ロールを追加する
- アカウント接続の表示
- スペースからのアカウントの削除 (内 CodeCatalyst)
- スペースの請求アカウントの設定

スペース CodeCatalyst にアカウントを追加 AWS アカウント することで、 認証済み を使用するように を設定できます。 CodeCatalyst スペース AWS アカウント に を追加することで、プロジェクトワークフローに AWS アカウント リソースと請求設定へのアクセスを許可できます。

を追加すると、このアカウントの使用 CodeCatalyst を に許可する接続 AWS アカウント が作成されます。追加した を使用して AWS アカウント 、次の操作を実行できます。

- CodeCatalyst スペースの請求を設定します。「Amazon CodeCatalyst 管理者ガイド」の「請求の管理」を参照してください。
- CodeCatalyst が IAM ロールを引き受けて AWS リソースにアクセスし、アカウントの AWS のサービス にデプロイできるようにします。接続された AWS リソースへのアクセスを許可する AWS アカウント を参照してください。

アカウント接続は、 で認証を完了することによって作成されます AWS アカウント。接続を作成したら、IAM ロールを追加して、ワークフローとプロジェクトが使用する接続をさらに設定します。

スペース AWS アカウント への の追加

CodeCatalyst コンソールと を使用して AWS Management Console 、スペースを に接続します AWS アカウント。

のスペース AWS アカウント に を追加する前に CodeCatalyst、次の前提条件を満たしていることを確認してください。

- AWS アカウント を作成し、接続するアカウントで IAM ロールを作成 AWS するためのアクセス許可を取得します。
- アカウント接続に関連付ける IAM ロールを作成します。これには、ロールのアクセス許可を持つ IAM ポリシーが含まれます。
- 接続を作成するスペースのスペース管理者ロールを取得します。 CodeCatalyst

トピック

ステップ 1: 接続リクエストを作成する

CodeCatalyst コンソールで接続リクエストを作成すると、認証を完了するために使用できる接続トークンが生成されます。

接続を作成するスペースには、スペース管理者またはパワーユーザーロールが必要です。CodeCatalyst 追加 AWS アカウント する の管理権限も必要です。

接続を作成する

1. で AWS Management Console、接続を作成するのと同じアカウントでログインしていることを確認します。
2. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
3. CodeCatalyst スペースに移動します。[Settings (設定)]、[AWS アカウント] の順に選択します。
4. を追加 AWS アカウントを選択します。
5. Amazon AWS アカウント に関連付ける CodeCatalystページの AWS アカウント ID に、スペースに接続するアカウントの 12 桁の ID を入力します。 AWS アカウント ID の検索については、AWS アカウント 「ID とそのエイリアス」を参照してください。
6. Amazon CodeCatalyst 表示名 に、アカウントの参照名を入力します。
7. (オプション) 接続の説明 に、アカウントとロール、またはロールが適用されるプロジェクトを選択するのに役立つアカウントの説明を入力します。
8. [関連付ける] AWS アカウント を選択します。
9. このページは、成功バナーが表示されるAWS アカウント 詳細ページに戻ります。

ステップ 2: アカウント接続リクエストを受け入れる

CodeCatalyst コンソールでリクエストを送信して に接続すると AWS アカウント、 AWS 管理者と協力して、提供された接続トークンで送信して接続リクエストを受け入れます。

アカウントの管理者権限があり、接続を作成するのと同じ AWS Management Console で AWS アカウント にサインインしていることを確認します。

接続リクエストを承認するには (コンソール)

1. で AWS Management Console、接続を作成するのと同じアカウントでログインしていることを確認します。
2. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
3. CodeCatalyst スペースに移動します。[Settings (設定)]、[AWS アカウント] の順に選択します。
4. AWS アカウント 詳細ページで、「」の「セットアップの完了」を選択します AWS Management Console。
5. で「Amazon CodeCatalyst スペースの検証」ページが開きます AWS Management Console。これは Amazon CodeCatalyst Spaces ページです。ページにアクセスするには、ログインが必要な場合があります。

   ページに直接アクセスするには、https://console.aws.amazon.com/codecatalyst/home/. の で Amazon CodeCatalyst Spaces にサインイン AWS Management Console します。

   検証トークンは検証トークン に自動的に入力されます。成功メッセージには、トークンが有効なトークンであることを示すメッセージが表示されます。
6. (オプション)「承認された有料利用枠」で「有料利用枠の承認 (スタンダード、エンタープライズ) 」を選択して、請求アカウントの有料利用枠を有効にします。

   > **Note**
   >
   > これにより、請求階層が有料階層にアップグレードされることはありません。ただし、これにより が構成 AWS アカウント され、 でいつでもスペースの請求階層を変更できます CodeCatalyst。有料利用枠はいつでも有効にできます。この変更を行わない場合、スペースは 無料利用枠のみを使用できます。
7. [スペースを確認] を選択します。

   アカウントがスペースに追加されたことを示すアカウント検証の成功メッセージが表示されます。

ステップ 3: 承認された接続を確認する

接続が承認されると、コンソールで接続と、その接続に追加した IAM ロールを表示できます。

承認された接続を確認するには

1. CodeCatalyst スペースに移動します。[Settings (設定)]、[AWS アカウント] の順に選択します。
2. アカウント接続は、作成日とともに一覧表示されます。
3. アカウント表示名を選択します。AWS アカウント 詳細ページが表示されます。

ステップ 4: 接続に IAM ロールを追加する

CodeCatalyst デプロイアクション用に設定された IAM ロールを使用している場合は、ロールをデプロイ環境に追加します。詳細については、「アカウント接続への IAM ロールの追加」を参照してください。

次のステップ: アカウント接続用の追加の IAM ロールを作成する

接続を作成したら、追加の IAM ロールを作成して追加できます。追加する IAM ロールは、ワークフローによって異なります。例えば、ビルドアクションには CodeCatalyst CodeCatalyst ビルドロールが必要です。

アカウントを接続するには、作成したロールの Amazon リソースネーム (ARN) が必要です。ここで説明するように、ロールの ARN をコピーします。IAM ロールの ARNs「Amazon リソースネーム (ARN)」を参照してください。

IAM ロール ARN にアクセスするには

1. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
2. ナビゲーションペインで Roles (ロール) を選択します。
3. 検索ボックスに、追加するロールの名前を入力します。
4. リストからロールを選択します。

   ロールの概要ページが表示されます。
5. 上部で、ロール ARN 値をコピーします。

アカウント接続への IAM ロールの追加

アカウント接続の作成には、 CodeCatalyst スペース内のプロジェクトで使用する IAM ロールの追加が含まれます。

> **Note**
>
> アカウント接続で IAM ロールを使用するには、 CodeCatalyst サービスプリンシパルを使用するように信頼ポリシーが更新されていることを確認します。

IAM ロールをアカウント接続に追加する (コンソール)

1. で AWS Management Console、管理するのと同じアカウントでログインしていることを確認します。
2. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
3. CodeCatalyst スペースに移動します。[Settings (設定)]、[AWS アカウント] の順に選択します。
4. アカウント接続の Amazon CodeCatalyst 表示名を選択し、 からロールの管理を選択します AWS Management Console。

   「Amazon CodeCatalyst スペースへの IAM ロールの追加」ページが表示されます。
5. 次のいずれかを行います。

   - 開発者ロールのアクセス許可ポリシーと信頼ポリシーを含むサービスロールを作成するには、IAM で CodeCatalyst 開発管理者ロールを作成するを選択します。ロールには一意の識別子`CodeCatalystWorkflowDevelopmentRole-`*`spaceName`*が付加された名前が付けられます。ロールとロールポリシーの詳細については、「」を参照してくださいCodeCatalystWorkflowDevelopmentRole-***`spaceName`*** サービスロールについて。

     開発ロールの作成 を選択します。
   - IAM で既に作成したロールを追加するには、既存の IAM ロールの追加 を選択します。既存の IAM ロールを選択 で、ドロップダウンリストからロールを選択します。

     [Add role] を選択します。

   ページが で開きます AWS Management Console。ページにアクセスするには、ログインが必要な場合があります。
6. Amazon CodeCatalyst スペースページのナビゲーションペインで、スペース を選択します。

   ページに直接アクセスするには、https://console.aws.amazon.com/codecatalyst/home/. の で Amazon CodeCatalyst Spaces にサインイン AWS Management Console します。
7. CodeCatalyst スペースに追加されたアカウントを選択します。接続ページが表示されます。

8. 接続ページの で使用できる IAM ロール CodeCatalystで、アカウントに追加された IAM ロールのリストを表示します。IAM ロールを に関連付けるを選択します CodeCatalyst。
9. IAM ロールの関連付けポップアップのロール ARN に、 CodeCatalyst スペースに関連付ける IAM ロールの Amazon リソースネーム (ARN) を入力します。

   目的 で、アカウント接続でロールを使用する方法を説明するロールの目的を選択します。ワークフローでアクションを実行するために使用するロールRUNNERに を指定します。別のサービスにアクセスするために使用するロールSERVICEには、 を指定します。

   複数の目的を指定できます。

   > **Note**
   >
   > ロール ARN の目的を選択する必要があります。

10. IAM ロールの関連付け を選択します。追加の IAM ロールについては、これらのステップを繰り返します。

デプロイ環境にアカウント接続と IAM ロールを追加する

Amazon ECS などの AWS リソースやデプロイ用の AWS Lambda リソースにアクセスするには、CodeCatalyst ビルドおよびデプロイアクションに、それらのリソースにアクセスするためのアクセス許可を持つ IAM ロールが必要です。Space 管理者または Power ユーザーロールを使用すると、CodeCatalyst アカウントをリソースが作成された AWS アカウント に接続できます。次に、IAM ロールをアカウント接続に追加します。デプロイアクションの場合は、IAM ロールを CodeCatalyst 環境に追加する必要があります。

プロジェクトのデプロイ環境で使用する IAM ロールを追加する必要があります。アカウント接続にロールを追加しても、ロールと 接続はプロジェクトデプロイ環境に追加されません。アカウント接続と IAM ロールをデプロイ環境に追加するには、アカウント接続とロールが「」で説明されているように作成されていることを確認してくださいステップ 4: 接続に IAM ロールを追加する。

次に、 CodeCatalyst コンソールの環境ページを使用して、アカウント接続と IAM ロールをプロジェクトのデプロイ環境に追加します。

> **Note**
>
> IAM ロールが IAM ロールを必要とする CodeCatalyst アクションに使用されている場合にのみ、IAM ロールを環境に追加します。ビルドアクションを含む、IAM ロールを必要とするすべてのワークフローアクションは、 CodeCatalyst 環境を使用する必要があります。

アカウント接続と IAM ロールをデプロイ環境に追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. アカウント接続と IAM ロールを追加するデプロイ環境を使用してプロジェクトに移動します。
3. CI/CD を展開し、環境 を選択します。
4. 環境を選択すると、追加のタブが表示されます。
5. AWS アカウント 接続タブを選択します。接続名 の下に、環境に追加されているアカウントがあれば一覧表示されます。
6. [関連付ける] AWS アカウント を選択します。<environment_name> AWS アカウント に関連付けるページが表示されます。
7. 接続 で、追加する IAM ロールを持つアカウント接続の名前を選択します。[関連付ける] を選択します。

アカウント接続の表示

接続のリストを表示したり、各接続の詳細を表示したりできます。

スペースの接続を管理するには、スペース管理者またはパワーユーザーロールが必要です。

CodeCatalyst スペースのすべての接続を表示するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 表示するアカウント接続があるスペースに移動します。
3. AWS アカウントタブを選択します。
4. AWS アカウント で、各接続のアカウント ID とステータスを含む、スペースのアカウント接続のリストを表示します。

アカウント接続の詳細を表示するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. CodeCatalyst スペースに移動します。[Settings (設定)]、[AWS アカウント] の順に選択します。
3. Amazon CodeCatalyst 表示名 で、接続名を選択します。詳細ページで、接続に関連付けられている IAM ロールのリストとその他の詳細を表示します。

スペースからのアカウントの削除 (内 CodeCatalyst)

不要になったアカウント接続を削除できます。この手順では、 CodeCatalyst を使用して、以前にスペースに追加したアカウント接続を削除します。これにより、アカウントがスペースの請求アカウントでない限り、スペースからアカウント接続が削除されます。

> **Important**
>
> アカウント接続が削除されると、再接続することはできません。必要に応じて、新しいアカウント接続を作成し、IAM ロールと環境を関連付けるか、請求を設定する必要があります。

CodeCatalyst スペースの使用量が無料利用枠を超えない場合でも、請求アカウントをスペースに指定する必要があります。指定された請求アカウントであるアカウントのスペースを削除する前に、スペースに別のアカウントを追加する必要があります。「Amazon CodeCatalyst 管理者ガイド」の「請求の管理」を参照してください。

> **Important**
>
> これらのステップを使用してアカウントを削除することはできますが、これはお勧めしません。アカウントは、 のワークフローをサポートするように設定することもできます CodeCatalyst。

スペースのアカウント接続を管理するには、スペース管理者またはパワーユーザーロールが必要です。

アカウント接続を削除するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。

2. CodeCatalyst スペースに移動します。[Settings (設定)]、[AWS アカウント] の順に選択します。
3. Amazon CodeCatalyst 表示名 で、削除するアカウント接続の横にあるセレクターを選択します。
4. [削除] AWS アカウント を選択します。フィールドに名前を入力して削除を確認し、削除を選択します。

   成功バナーが表示され、アカウント接続が接続のリストから削除されます。

スペースの請求アカウントの設定

CodeCatalyst スペースの使用量が無料利用枠を超えない場合でも、請求アカウントをスペースに指定する必要があります。

請求アカウントを設定するには、「 CodeCatalyst 管理者ガイド」の「請求」を参照してください。の ページを使用して、スペースに追加されたアカウント CodeCatalyst AWS を削除できます。この手順では、管理している特定のアカウントの管理権限を使用して、 の Amazon CodeCatalyst Spaces ページにサインイン AWS Management Console し、スペース AWS アカウント から を削除します。 CodeCatalyst スペースの指定された請求アカウントであるアカウントを削除するには、まず新しい請求アカウントを指定してください。

削除されたアカウントは後で再度追加できますが、アカウントとスペースの間に新しい接続を作成する必要があります。IAM ロールは、追加したアカウントに再関連付けする必要があります。

接続されたアカウントの IAM ロールの設定

に追加するアカウントのロールを AWS Identity and Access Management (IAM) で作成します CodeCatalyst。請求アカウントを追加する場合は、ロールを作成する必要はありません。

では AWS アカウント、 AWS アカウント スペースに追加する のロールを作成するアクセス許可が必要です。IAM リファレンスやポリシーの例など、IAM ロールとポリシーの詳細については、「」を参照してくださいIdentity and Access Management と Amazon CodeCatalyst。で使用される信頼ポリシーとサービスプリンシパルの詳細については CodeCatalyst、「」を参照してください CodeCatalyst 信頼モデルを理解する。

では CodeCatalyst、スペース管理者ロールを使用してサインインし、スペースにアカウント (および該当する場合はロール) を追加する手順を完了する必要があります。

次のいずれかの方法を使用して、アカウント接続にロールを追加できます。

- ロールのアクセス許可ポリシーと信頼ポリシーを含むサービスCodeCatalystWorkflowDevelopmentRole-*spaceName*ロールを作成するには、「」を参照してくださいCodeCatalystWorkflowDevelopmentRole-*spaceName* ロール。
- ロールを作成し、設計図からプロジェクトを作成するためのポリシーを追加する例については、「」を参照してくださいIAM ロールの作成と信頼ポリシーの使用 CodeCatalyst 。
- IAM ロールの作成時に使用するロールポリシーのサンプルのリストについては、「」を参照してくださいIAM ロールを使用してプロジェクト AWS リソースへのアクセスを許可する。
- ワークフローアクションのロールを作成する詳細な手順については、次のように、そのアクションのワークフローチュートリアルを参照してください。
  - チュートリアル: Amazon S3 にアーティファクトをアップロードする
  - チュートリアル: を使用してサーバーレスアプリケーションをデプロイする AWS CloudFormation
  - チュートリアル: Amazon ECS にアプリケーションをデプロイする
  - チュートリアル: ワークフローでアクションを使用して GitHubコードをリントする

トピック

- CodeCatalystWorkflowDevelopmentRole-spaceName ロール
- AWSRoleForCodeCatalystSupport ロール
- IAM ロールの作成と信頼ポリシーの使用 CodeCatalyst

CodeCatalystWorkflowDevelopmentRole-*spaceName* ロール

開発者ロールは、IAM でワンクリックロールとして作成します。アカウントを追加するスペースには、スペース管理者またはパワーユーザーロールが必要です。追加 AWS アカウント する の管理権限も必要です。

以下の手順を開始する前に、 CodeCatalyst スペースに追加するのと同じアカウント AWS Management Console で にログインする必要があります。そうしないと、コンソールは不明なアカウントエラーを返します。

を作成して追加するには CodeCatalyst CodeCatalystWorkflowDevelopmentRole-*spaceName*

1. CodeCatalyst コンソールで を開始する前に、 を開き AWS Management Console、 AWS アカウント スペースに対して同じ でログインしていることを確認します。
2. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。

3. CodeCatalyst スペースに移動します。[Settings (設定)]、[AWS アカウント] の順に選択します。
4. ロール AWS アカウント を作成する のリンクを選択します。AWS アカウント 詳細ページが表示されます。
5. からロールの管理を選択します AWS Management Console。

   で「Amazon CodeCatalyst スペースへの IAM ロールの追加」ページが開きます AWS Management Console。これは Amazon CodeCatalyst スペーススページです。ページにアクセスするには、ログインが必要な場合があります。
6. IAM で CodeCatalyst 開発管理者ロールの作成を選択します。このオプションは、開発ロールのアクセス許可ポリシーと信頼ポリシーを含むサービスロールを作成します。ロールには という名前が付けられますCodeCatalystWorkflowDevelopmentRole-*spaceName*。ロールとロールポリシーの詳細については、「」を参照してくださいCodeCatalystWorkflowDevelopmentRole-***spaceName*** サービスロールについて。

   > **Note**
   >
   > このロールはデベロッパーアカウントでのみ使用が推奨され、`AdministratorAccess` AWS 管理ポリシーを使用して、この で新しいポリシーとリソースを作成するためのフルアクセスを付与します AWS アカウント。
7. 開発ロールの作成 を選択します。
8. 接続ページの で使用できる IAM ロールで CodeCatalyst、アカウントに追加された IAM `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`*ロールのリストでロールを表示します。
9. スペースに戻るには、「Amazon に移動 CodeCatalyst」を選択します。

AWSRoleForCodeCatalystSupport ロール

サポートロールは、IAM でワンクリックロールとして作成します。アカウントを追加するスペースには、スペース管理者またはパワーユーザーロールが必要です。追加 AWS アカウント する の管理権限も必要です。

以下の手順を開始する前に、 CodeCatalyst スペースに追加するのと同じアカウント AWS Management Console で にログインする必要があります。そうしないと、コンソールは不明なアカウントエラーを返します。

を作成して追加するには CodeCatalyst AWSRoleForCodeCatalystSupport

1. CodeCatalyst コンソールで を開始する前に、 を開き AWS Management Console、 AWS アカウント スペースに対して同じ でログインしていることを確認します。
2. CodeCatalyst スペースに移動します。[Settings (設定)]、[AWS アカウント] の順に選択します。
3. ロール AWS アカウント を作成する のリンクを選択します。AWS アカウント 詳細ページが表示されます。
4. からロールの管理を選択します AWS Management Console。

   で「Amazon CodeCatalyst スペースへの IAM ロールの追加」ページが開きます AWS Management Console。これは Amazon CodeCatalyst Spaces ページです。ページにアクセスするには、サインインが必要な場合があります。
5. CodeCatalyst スペースの詳細 で、 CodeCatalyst サポートロールの追加 を選択します。このオプションは、プレビュー開発ロールのアクセス許可ポリシーと信頼ポリシーを含むサービスロールを作成します。ロールには一意の識別子AWSRoleForCodeCatalystSupportが付加された名前が付けられます。ロールとロールポリシーの詳細については、「」を参照してくださいAWSRoleForCodeCatalystSupport サービスロールについて。
6. CodeCatalyst サポート用のロールを追加 ページで、デフォルトを選択したまま、ロールの作成を選択します。
7. で使用できる IAM ロール CodeCatalystで、アカウントに追加された IAM `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`*ロールのリストでロールを表示します。
8. スペースに戻るには、「Amazon に移動 CodeCatalyst」を選択します。

IAM ロールの作成と信頼ポリシーの使用 CodeCatalyst

で CodeCatalyst AWS アカウント 接続で使用する IAM ロールは、ここで提供される信頼ポリシーを使用するように設定する必要があります。以下のステップを使用して IAM ロールを作成し、 でブループリントからプロジェクトを作成できるようにするポリシーをアタッチします CodeCatalyst。

別の方法として、ロールのアクセス許可ポリシーと信頼ポリシーを含むサービスCodeCatalystWorkflowDevelopmentRole-*spaceName*ロールを作成できます。詳細については、「アカウント接続への IAM ロールの追加」を参照してください。

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。

2. [ロール]、[ロールの作成] の順に選択します。
3. カスタム信頼ポリシー を選択します。
4. カスタム信頼ポリシーフォームに、次の信頼ポリシーを貼り付けます。

```
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
             "Principal": {
                "Service": [
                    "codecatalyst-runner.amazonaws.com",
                    "codecatalyst.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:codecatalyst:::space/spaceId/project/
*"
                }
            }
        }
    ]
```

5. [次へ] をクリックします。
6. 「アクセス許可の追加」で、IAM で既に作成したカスタムポリシーを検索して選択します。
7. [次へ] をクリックします。
8. ロール名 には、ロールの名前を入力します。例: `codecatalyst-project-role`
9. [ロールを作成] を選択します。
10. ロール Amazon リソースネーム (ARN) をコピーします。この情報は、ロールをアカウント接続または環境に追加するときに提供する必要があります。

ユーザーにスペース許可を付与する

スペースに参加するユーザーのロールを表示、追加、削除、または変更することで、スペースのメンバーを管理できます。

このガイドの情報は、 AWS Builder ID ユーザーをサポートする のスペースでユーザー CodeCatalyst を招待および管理するために提供されています。ID フェデレーションをサポートするスペースを設定および管理するためのステップの詳細については、「Amazon CodeCatalyst 管理者ガイド」の CodeCatalyst「スペースのセットアップと管理」を参照してください。

スペース内のメンバーの表示

表示名、エイリアス、スペースのロールに関する情報など、スペース内のユーザーを表示できます。スペース内のメンバーには 3 つのロールがあります。

- スペース管理者 – このロールには CodeCatalyst、プロジェクトの作成を含む、 のすべてのアクセス許可があります。このロールは、スペース内のすべてのプロジェクトへのアクセスなど、スペースのあらゆる側面を管理する必要があるユーザーにのみ割り当てます。

  このロールを後で変更するには、まずユーザーを削除します。詳細については、「スペース管理者ロール」を参照してください。
- パワーユーザー – このロールは Amazon CodeCatalyst スペースで 2 番目に強力なロールですが、スペース内のプロジェクトにはアクセスできません。これは、スペースにプロジェクトを作成し、スペースのユーザーとリソースを管理するのに役立つ必要があるユーザー向けに設計されています。詳細については、「パワーユーザーロール」を参照してください。
- 制限付きアクセス – このロールは、スペース内のプロジェクトへの招待を受け入れてスペースに参加するユーザーにデフォルトで割り当てられます。プロジェクトメンバーには、プロジェクト内のロールが割り当てられます。プロジェクトメンバーの管理については、「」を参照してくださいユーザープロジェクトのアクセス許可の付与。

Space 管理者テーブルには、Space 管理者ロールを持つユーザーが表示されます。これらのユーザーは、スペース内のすべてのプロジェクトに自動的に (暗黙的に) 割り当てられ、プロジェクトにロールがないため、スペースメンバーに表示されません。

Space members テーブルには、スペース管理者ロールを持たない状態で、プロジェクトにロールを持つスペース内のすべてのメンバーが表示されます。

ユーザーは、ユーザーが でスペース管理者ロールを持っているかどうかに基づいて CodeCatalyst 次のように表示されます。

- Space 管理者ロールを持つユーザーが後でプロジェクトの招待を受け入れ、ロールはスペースの下のスペースメンバーテーブルやプロジェクトのメンバーテーブルに表示されません。これらは引き続き両方の場所のスペース管理者テーブルに表示されます。各プロジェクトでは、スペース管理

者ロールを持つすべてのユーザーが、そのプロジェクトのプロジェクトスペース管理者テーブルに表示されます。

- プロジェクトロールに参加するためのプロジェクト招待を受け入れるユーザーは、制限付きアクセスロールを持つスペースに追加されます。ユーザーのロールが後で Space 管理者ロールに変更されても、 は Space メンバーテーブルから Space 管理者テーブルに移動します。プロジェクトの下で、ユーザーはプロジェクトメンバーテーブルからスペース管理者テーブルに移動します。

スペース内のユーザーとロールを表示するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 自分のスペースに移動します。

   > **Tip**
   >
   > 複数のスペースに属している場合は、上部のナビゲーションバーでスペースを選択します。

3. 設定 を選択し、メンバー を選択します。

   スペースのメンバーであるユーザーは、スペースメンバーテーブルに表示されます。

   > **Tip**
   >
   > Space 管理者ロールがある場合は、直接招待されたプロジェクトを表示できます。プロジェクトのプロジェクト設定に移動し、プロジェクト を選択します。

   ステータス列では、有効な値は次のとおりです。

   - 招待済み – 招待 CodeCatalyst を送信しましたが、ユーザーはまだ承諾または拒否していません。
   - メンバー – ユーザーは招待を受け入れました。

ユーザーをスペースに直接招待する

ユーザーを CodeCatalyst スペースに直接招待できます。これは、スペース管理者またはパワーユーザーロールを割り当てて、そのユーザーをスペースの管理に招待する場合に便利です。これらのロー

ルの 1 つを他のユーザーに割り当てると、これらのユーザーをプロジェクトに招待することなく、より多くのユーザーに領域を管理する責任を分散できます。

> **Note**
>
> メンバーを招待するには、スペース管理者またはパワーユーザーロールが必要です。

Space 管理者テーブルには、Space 管理者ロールを持つユーザーが表示されます。これらのユーザーは、スペース内のすべてのプロジェクトに自動的に (暗黙的に) 割り当てられ、プロジェクトにロールがないため、スペースメンバーテーブルに表示されません。

プロジェクトの招待を受け入れるメンバーは、デフォルトでスペースに追加されます。プロジェクトメンバーテーブルには、プロジェクト内のロールを持つスペース内のすべてのメンバーが表示されます。

招待を受け入れて初めてサインインする方法については、「」を参照してくださいのセットアップとへのサインイン CodeCatalyst。

ユーザーをスペースに招待するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 自分のスペースに移動します。
3. 設定 を選択し、メンバー を選択します。
4. 招待を選択します。
5. スペースへの参加を招待するユーザーの E メールを入力します。ロール で、そのユーザーをスペースに割り当てるロールを選択します。
6. 招待を選択する

スペースの招待をキャンセルする

最近送信したスペースへの参加の招待をキャンセルし、まだ承諾されていない場合は、キャンセルできます。

スペースの招待を管理するには、スペース管理者またはパワーユーザーロールが必要です。

スペースメンバーの招待をキャンセルするには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。

2. 自分のスペースに移動します。

> **Tip**
>
> 複数のスペースに属している場合は、上部のナビゲーションバーでスペースを選択します。

3. 設定 を選択し、メンバー を選択します。
4. メンバーのステータスが招待済み であることを確認します。

> **Note**
>
> キャンセルできるのは、まだ承諾されていない招待のみです。

5. 招待されたメンバーを含む行の横にあるオプションを選択し、招待をキャンセル を選択します。
6. 確認ウィンドウが表示されます。「招待をキャンセルする」を選択して確認します。

スペースメンバーのロールの変更

スペースのメンバーに割り当てられたロールを変更できます。スペース内のユーザーのロールを変更するには、スペース管理者ロールが必要です。

Space 管理者テーブルには、Space 管理者ロールを持つユーザーが表示されます。これらのユーザーは、スペース内のすべてのプロジェクトに自動的に (暗黙的に) 割り当てられるため、スペースメンバーテーブルには表示されません。

スペース内のユーザーのロールを変更するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 自分のスペースに移動します。

> **Tip**
>
> 複数のスペースに属している場合は、上部のナビゲーションバーでスペースを選択します。

3. 設定 を選択し、メンバー を選択します。

4. Space members テーブルで、ロールを変更するユーザーを選択します。ロールの変更 を選択します。

スペースメンバーの削除

スペースリソースにアクセスする必要がない場合は、スペースのメンバーを削除できます。スペースからメンバーを削除するには、スペース管理者ロールが必要です。

Space 管理者テーブルには、Space 管理者ロールを持つユーザーが表示されます。これらのユーザーは、スペース内のすべてのプロジェクトに自動的に (暗黙的に) 割り当てられ、プロジェクトにロールがないため、スペースメンバーテーブルに表示されません。このテーブルでは、スペースのメンバーを直接削除することしかできません。

プロジェクトメンバーテーブルからユーザーを削除するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 自分のスペースに移動します。

> **Tip**
>
> 複数のスペースに属している場合は、上部のナビゲーションバーでスペースを選択します。

3. 設定 を選択し、メンバー を選択します。
4. プロジェクトメンバーテーブルでユーザーを選択します。[削除] を選択します。

> **Note**
>
> スペースからメンバーを削除すると、そのスペース内のすべてのプロジェクトからユーザーが削除され、それらのプロジェクトのリソースに関連付けられたアクセス許可も削除されます。

Space 管理者ロールを持つユーザーのロールの削除または変更

スペースのスペース管理者ロールを持つユーザーのロールを削除または変更できます。

スペース管理者ロールを持つユーザーをスペースから削除するには、スペース管理者ロールが必要です。Space 管理者ロールを持つユーザーのロールを変更すると、基本的にそのユーザーは Space 管理者テーブルから削除されます。そのユーザーがスペース内のプロジェクトにプロジェクトロールを持っていない場合、ユーザーからスペース管理者ロールを削除すると、そのユーザーはスペースから削除されます。

> **Note**
>
> Space 管理者ロールを持つユーザーは、自分自身を削除することはできません。Space 管理者ロールを持つ他のユーザーに連絡してください。

Space 管理者ロールを持つユーザーを Spaceメンバーテーブルから削除するには

> **Note**
>
> プロジェクトに明示的に追加されていないユーザーには、プロジェクトロール (プロジェクト管理者または寄稿者) はありません。スペース管理者ロールがユーザーの唯一のロールである場合、ユーザーはスペースから完全に削除されます。

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. Space 管理者ロールを持つユーザーのロールを削除または変更するスペースに移動します。
3. 設定 を選択し、メンバー を選択します。
4. メンバーのリストの招待ステータスを表示し、そのリストにスペースへの未許可の保留中の招待 (招待済み のステータス) が含まれていないことを確認します。

   > **Important**
   >
   > Space 管理者ロールを持つユーザーを削除する前に、保留中の招待が開始されていないことを確認する必要があります。

5. [メンバー] タブを選択します。スペース管理者テーブルでユーザーを選択し、 の削除を選択します。

   メンバーの削除ダイアログボックスで、次のいずれかを実行します。

- ユーザーの Space 管理者ロールのみを削除するオプションを選択します。[削除] を選択します。

> **Important**
>
> ユーザーに他のロールが割り当てられていない場合、スペース管理者からロールを変更すると、そのユーザーはスペースから削除されます。

- スペース管理者ロールを持つユーザーをスペースとそのすべてのプロジェクトから削除するオプションを選択します。[削除] を選択します。

6. メンバー タブを更新します。ユーザーは、プロジェクトロールを通じてユーザーがメンバーシップを持っていたプロジェクトのプロジェクトメンバーのリストに自動的に追加されます。Space 管理者ロールがユーザーの唯一のロールである場合、ユーザーはスペースから完全に削除されます。

チームを使用したスペースアクセスの許可

スペースを作成したら、チームを追加できます。チームを使用すると、ユーザーをグループ化して、ユーザーが でアクセス許可を共有し、プロジェクト、問題の追跡、ロール、リソースを管理できます CodeCatalyst。

チームを管理するには、スペース管理者ロールが必要です。

チームも のプロジェクト/スペースレベルで管理されます CodeCatalyst。スペース/プロジェクトのチームの詳細については、「」を参照してくださいチームを使用したスペースアクセスの許可。

トピック

- チームの作成
- チームの表示
- チームにスペースロールを付与する
- スペースレベルでチームにプロジェクトロールを付与する
- ユーザーをチームに直接追加する
- チームからユーザーを直接削除する
- SSO グループをチームに追加する
- チームの削除

チームの作成

チームには、スペース内の Power user などのロール許可があります。チームは、プロジェクト内のプロジェクト管理者 などのプロジェクトアクセス許可を持つこともできます。チームは、プロジェクトごとに異なるロールを持つ多くのプロジェクトに関連付けることができます。チームメンバーが Builder ID スペースの個々のユーザーであるか、ID AWS フェデレーションをサポートするスペースの SSO グループであるかのいずれかであるチームを管理できます。

スペースユーザーとプロジェクトユーザーのメンバーページで、ユーザーは複数のロールを持つことができます。複数のロールを持つユーザーには、複数のロールがある場合にインジケータが表示され、最もアクセス許可の大きいロールが最初に表示されます。

> **Note**
>
> スペースが ID フェデレーションをサポートしている場合は、IAM Identity Center で SSO ユーザーまたは SSO グループをセットアップしておく必要があります。

チームメンバーの管理方法は、ユーザーの追加と削除の方法によって異なります。チームメンバーを管理するには、次の 2 つのオプションがあります。

- ユーザーを直接追加する — ユーザーを個別に追加または削除します。例えば、IAM Identity Center で既にセットアップされている AWS Builder ID ユーザーまたは SSO ユーザーを選択して、チームにユーザーを追加します。Builder ID ユーザーまたは AWS SSO ユーザーを直接追加してチームメンバーを管理することを選択した場合、SSO グループを使用するオプションは使用できなくなります。
- SSO グループを使用する — IAM Identity Center で既にセットアップされている SSO グループを通じてチームメンバーを管理します。SSO グループ を使用してチームメンバーを管理することを選択した場合、ユーザーを直接追加するオプションは使用できなくなります。

チームを管理するには、スペース管理者ロールが必要です。

チームを作成するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 自分のスペースに移動します。設定 を選択し、チーム を選択します。
3. チームの作成 を選択します。

4. チーム名 に、チームのわかりやすい名前を入力します。

> **Note**
>
> チーム名はスペース内で一意である必要があります。

(オプション) チームの説明 に、チームの説明を入力します。

5. スペースロール で、チーム CodeCatalyst に割り当てる で使用可能なスペースロールのリストからロールを選択します。ロールはチームのすべてのメンバーに継承されます。

   - スペース管理者 - 詳細については、「」を参照してくださいスペース管理者ロール。
   - 制限付きアクセス - 詳細については、「」を参照してください制限付きアクセスロール。
   - パワーユーザー - 詳細については、「」を参照してくださいパワーユーザーロール。

6. チームメンバーシップ で、次のいずれかを選択して、メンバーをチームに追加する方法を選択します。

   - メンバーを直接追加を選択して、ユーザーを個別に管理します。これには、スペースに AWS Builder ID ユーザーを追加したり、ID フェデレーションをサポートするスペースに SSO ユーザーを追加したりすることが含まれます。
   - SSO グループを使用 を選択して、IAM Identity Center で既にセットアップした SSO グループを選択します。

     SSO グループ で、追加するグループの横にあるボックスを選択します。最大 5 つの SSO グループを追加できます。

> **Note**
>
> 後で変更することはできません。Builder ID ユーザーまたは AWS SSO ユーザーを直接追加してチームメンバーを管理することを選択した場合、SSO グループを使用するオプションは使用できなくなります。SSO グループ を使用してチームメンバーを管理することを選択した場合、ユーザーを直接追加するオプションは使用できなくなります。

7. [作成] を選択します。

> **Note**
>
> SSO グループの使用を選択した場合、SSO グループのユーザーはチームの作成時にプルされないことに注意してください。ユーザーがリストに表示される CodeCatalyst 前に、 にサインインしている必要があります。

チームの表示

では CodeCatalyst、チームのプロジェクトとロールを表示できます。メンバーページで、プロジェクトロールとユーザーのリストを表示できます。SSO グループタイプのチームの場合、チームに関連付けられた SSO グループのリストも表示できます。

チームを表示するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 自分のスペースに移動します。設定 を選択し、チーム を選択します。
3. スペースロール で、このスペースのチームに割り当てられたロールを表示します。
4. プロジェクトロール タブで、チームがメンバーとして追加されたスペース (AWS ビルダー ID スペースのみ) 内の各プロジェクトのチームに割り当てられたプロジェクトと CodeCatalyst プロジェクトロールを表示します。
5. メンバー タブで、チームに割り当てられたメンバーのリストを表示します。
6. SSO Groups タブで、チームに割り当てられた SSO グループのリストを表示します (ID フェデレーションのみをサポートするスペースの場合) 。

チームにスペースロールを付与する

チームは、 のプロジェクトへのチームアクセスを許可および管理できるようにユーザーをグループ化する方法です CodeCatalyst。例えば、チームを使用してユーザーのロールとアクセス許可をすばやく管理できます。これにより、チームはユーザーのスペースを管理できます。

チームには、スペース内の Power user などのロール許可があります。チームのスペースロールは変更できますが、チームのすべてのメンバーがこれらのアクセス許可を継承することに注意してください。

チームを管理するには、スペース管理者ロールが必要です。

チームのスペースロールの変更

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 自分のスペースに移動します。設定 を選択し、チーム を選択します。
3. アクション で、スペースロールの変更 を選択します。スペースロールは、次のいずれかに変更できます。これにより、チームのすべてのメンバーのロールが変更されます。
   - スペース管理者 - 詳細については、「」を参照してくださいスペース管理者ロール。
   - 制限付きアクセス - 詳細については、「」を参照してください制限付きアクセスロール。
   - パワーユーザー - 詳細については、「」を参照してくださいパワーユーザーロール。
4. [保存] を選択します。

スペースレベルでチームにプロジェクトロールを付与する

のチームは、チームメンバーがプロジェクト管理者 などのロール許可をプロジェクトに持つことができるという点でユーザーと CodeCatalyst 似ています。ロールの変更がチームに適用され、チームのすべてのメンバーがそれらのアクセス許可を継承します。チームに自動的に付与されるプロジェクトごとに 1 つのロールを選択できます。

チームを管理するには、スペース管理者ロールが必要です。

プロジェクトロールを追加または変更するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 自分のスペースに移動します。設定 を選択し、チーム を選択します。
3. プロジェクトロールタブを選択します。
4. ロールを変更するには、このリストのプロジェクトの横にあるセレクターを選択し、ロールの変更を選択します。ロールを追加するには、「プロジェクトロールの追加」を選択します。プロジェクト で追加するプロジェクトを選択し、ロール でロールを選択します。使用可能なプロジェクトロールのいずれかを選択します。
   - プロジェクト管理者 - 詳細については、「」を参照してくださいプロジェクト管理者ロール。
   - Contributor - 詳細については、「」を参照してください寄稿者ロール。
   - レビューワー - 詳細については、「」を参照してくださいレビューワーロール。
   - 読み取り専用 - 詳細については、「」を参照してください読み取り専用ロール。
5. [保存] を選択します。

プロジェクトロールを削除するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 自分のスペースに移動します。設定 を選択し、チーム を選択します。
3. プロジェクトロールタブを選択します。
4. 削除するロールを選択します。

> **Important**
>
> チームからロールを削除すると、チーム内のすべてのユーザーに関連付けられたアクセス許可が削除されます。

5. [保存] を選択します。

ユーザーをチームに直接追加する

チームメンバーをチームに追加できます。ユーザーを追加すると、新しいユーザーはチームの既存のすべてのロールからアクセス許可を継承します。

Builder ID ユーザーサポートまたは ID AWS フェデレーション用にスペースを設定するかどうかにかかわらず、ユーザーを直接追加するようにスペースを設定できます。

> **Note**
>
> SSO グループを使用してチームメンバーを管理するようにスペースを設定する場合、ユーザーの追加を直接使用するオプションは使用できません。SSO グループを使用するには、「」を参照してくださいSSO グループをチームに追加する。

チームを管理するには、スペース管理者ロールが必要です。

ユーザーを直接追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 自分のスペースに移動します。設定 を選択し、チーム を選択します。
3. [メンバー] タブを選択します。
4. メンバーの追加 を選択します。

> **Note**
>
> チームに追加されるユーザーは、すでにスペースのメンバーである必要があります。スペースのメンバーではないチームメンバーを追加または招待することはできません。

5. ドロップダウンフィールドでユーザーを選択し、保存 を選択します。IAM Identity Center で既にセットアップされている AWS Builder ID ユーザーまたは SSO ユーザーを選択します。

チームからユーザーを直接削除する

チームからチームメンバーを削除できます。すべてのアクセス許可は、ユーザーによって継承されなくなります。後でユーザーをチームに追加し直すことができます。

> **Note**
>
> チームメンバーを削除すると、そのユーザーに関連付けられたアクセス許可が、スペース内のすべてのプロジェクトとリソースから削除されます。

チームを管理するには、スペース管理者ロールが必要です。

チームメンバーを削除するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 自分のスペースに移動します。設定 を選択し、チーム を選択します。
3. [メンバー] タブを選択します。
4. 削除するユーザーの横にあるセレクターを選択し、 の削除を選択します。
5. 入力フィールドに remove と入力し、 削除 を選択します。

SSO グループをチームに追加する

スペースが IAM Identity Center で管理されている SSO ユーザーとグループを含むスペースとして設定されている場合は、スペースを別のチームとして参加させる SSO グループを追加できます。

> **Note**
>
> Builder ID ユーザーまたは AWS SSO ユーザーを直接追加してチームメンバーを管理することを選択した場合、SSO グループを使用するオプションは使用できません。ユーザーを直接追加するには、「」を参照してくださいユーザーをチームに直接追加する。

チームを管理するには、スペース管理者ロールが必要です。

SSO グループをチームとして追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. スペースのページで、チーム を選択します。SSO グループタブを選択します。
3. 追加する SSO グループを選択します。最大 5 つの SSO グループを追加できます。

チームの削除

不要になったチームを削除できます。

> **Note**
>
> チームを削除すると、スペース内のすべてのプロジェクトとリソースからすべてのチームメンバーに関連付けられたアクセス許可が削除されます。

チームを管理するには、スペース管理者ロールが必要です。

チームを削除する

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 自分のスペースに移動します。設定 を選択し、チーム を選択します。
3. アクション で、チームの削除 を選択します。これにより、チーム全体のロールが変更されます。
4. [削除] をクリックします。

マシンリソースのスペースアクセスを許可する

マシンリソースは、 内のプロジェクトまたはスペースに対するアクセス許可 CodeCatalyst が付与されている 内の特定のリソースです CodeCatalyst。

> **Note**
>
> マシンリソースという用語は、Amazon EC2 インスタンスなどのクラウドインフラストラクチャを指すのではなく、スペースまたはプロジェクトのアクセス許可を持つ設計図またはワークフローリソースを指すためのものです。

マシンリソースは、 SSO CodeCatalyst 経由で にアクセスするときに、承認されたリソースからの ID を表します。マシンリソースは、ブループリントやワークフロー など、スペース内のリソースにアクセス許可を付与するために使用されます。スペース内のマシンリソースを表示したり、スペースのマシンリソースを有効または無効にしたりできます。例えば、マシンリソースを無効にしてアクセスを管理し、後で再度有効にすることができます。

これらのオペレーションは、マシンリソースを取り消したり無効にしたりする必要がある場合に、マシンリソースで使用できます。例えば、認証情報が侵害された可能性がある場合は、マシンリソースを無効にできます。通常、これらのオペレーションを使用する必要はありません。

このページを表示し、スペースレベルでマシンリソースを管理するには、スペース管理者ロールが必要です。

マシンリソースは、 のプロジェクトレベルでも管理されます CodeCatalyst。プロジェクトのチームの詳細については、「」を参照してくださいマシンリソースのスペースアクセスを許可する 。

トピック

- マシンリソースのスペースアクセスの表示
- マシンリソースのスペースアクセスを無効にする
- マシンリソースのスペースアクセスの有効化

マシンリソースのスペースアクセスの表示

スペースで使用されているマシンリソースのリストを表示できます。

マシンリソースを管理するには、スペース管理者ロールが必要です。

マシンリソースを表示するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. スペースに移動し、設定 を選択します。マシンリソース を選択します。
3. ドロップダウンで、ワークフローアクションを選択して、ワークフローのマシンリソースのみを表示します。ブループリントを選択すると、ブループリントのマシンリソースのみが表示されます。

   フィルターフィールドを使用して名前をフィルタリングすることもできます。

マシンリソースのスペースアクセスを無効にする

スペースで使用されているマシンリソースを無効にすることもできます。

> **Important**
>
> マシンリソースを無効にすると、スペース内のすべての関連するブループリントまたはワークフローに対するすべてのアクセス許可が削除されます。

マシンリソースを管理するには、スペース管理者ロールが必要です。

マシンリソースを無効にするには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. スペースに移動し、設定 を選択します。マシンリソース を選択します。
3. 次のいずれかを選択します 。

   > **Important**
   >
   > マシンリソースを無効にすると、スペース内のすべての関連するブループリントまたはワークフローに対するすべてのアクセス許可が削除されます。

   - 個別に無効にするには、無効にする 1 つ以上のマシンリソースの横にあるセレクターを選択します。無効化 を選択し、このリソース を選択します。
   - すべてのリソースを無効にするには、 を無効化 を選択し、すべてのリソース を選択します。

- すべてのワークフローアクションを無効にするには、 を無効にする を選択し、次にすべてのワークフローアクション を選択します。
- すべてのブループリントを無効にするには、 を無効にする を選択し、すべてのブループリント を選択します。

マシンリソースのスペースアクセスの有効化

スペースで使用中で無効になっているマシンリソースを有効にすることができます。

マシンリソースを管理するには、スペース管理者ロールが必要です。

マシンリソースを有効にするには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. スペースに移動し、設定 を選択します。マシンリソース を選択します。
3. 次のいずれかを選択します 。

   - を個別に有効にするには、有効にする 1 つ以上のマシンリソースの横にあるセレクターを選択します。を有効化 を選択し、このリソース を選択します。
   - すべてのリソースを有効にするには、「 を有効にする」を選択し、「すべてのリソース」を選択します。
   - すべてのワークフローアクションを有効にするには、「 を有効にする」を選択し、「すべてのワークフローアクション」を選択します。
   - すべてのブループリントを有効にするには、 を有効にする を選択し、すべてのブループリント を選択します。

スペースの開発環境の管理

すべての開発環境は、スペース内のプロジェクトの一部として作成されます。スペースメンバーは、ソースリポジトリレベルでプロジェクト内に独自の開発環境を作成できます。その後、スペース管理者は Amazon CodeCatalyst コンソールを使用して、スペースメンバーに代わって開発環境を表示、編集、削除、停止できます。つまり、スペース管理者は開発環境をスペースレベルで管理します。

開発環境の管理に関する考慮事項

- 「設定」の「開発環境」ページを表示し、スペースレベルで開発環境を管理するには、スペース管理者ロールが必要です。

- スペースメンバーは、 CodeCatalyst アカウントを通じてプロジェクトで作成する開発環境を管理します。開発環境をスペース管理者として管理する場合、スペースメンバーに代わってこれらのリソースを維持します。
- 開発環境は、デフォルトで特定のコンピューティングおよびストレージ設定になります。設定のアップグレードの請求と料金については、Amazon の CodeCatalyst 料金ページ を参照してください。

実行中のインスタンスの停止、デフォルトのコンピューティング設定、コンピューティングのアップグレード、コストの発生、タイムアウトの設定など、開発環境に関するその他の考慮事項については、「」を参照してください[で開発環境を使用してコードを記述および変更する CodeCatalyst]。

トピック

スペースの開発環境の表示

スペース内のすべての開発環境のタイプ、ステータス、詳細を表示できます。開発環境の作成と実行の詳細については、「」を参照してください[開発環境の作成]。

このページを表示し、スペースレベルで開発環境を管理するには、スペース管理者ロールが必要です。

スペース内の開発環境を表示するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. CodeCatalyst スペースに移動します。

> **Tip**
>
> 複数のスペースに属している場合は、上部のナビゲーションバーでスペースを選択します。

3. 設定 を選択し、開発環境 を選択します。

このページには、スペース内のすべての開発環境が一覧表示されます。リソース名、該当する場合はリソースエイリアス、IDE のタイプ、デフォルトまたは設定されたコンピューティングとストレージ、および各開発環境に対して設定されたタイムアウトを表示できます。

スペースの開発環境の編集

アイドル状態の開発環境の実行を停止するために、タイムアウト時間の設定など、開発環境の設定を編集できます。開発環境の編集の詳細については、「」を参照してください開発環境の編集。

このページを表示し、スペースレベルで開発環境を管理するには、スペース管理者ロールが必要です。

スペース内の開発環境を編集するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. CodeCatalyst スペースに移動します。

> **Tip**
> 複数のスペースに属している場合は、上部のナビゲーションバーでスペースを選択します。

3. 設定 を選択し、開発環境 を選択します。
4. 管理する開発環境の横にあるセレクターを選択します。[編集] を選択します。
5. 開発環境のコンピューティングまたは非アクティブタイムアウトに必要な変更を加えます。
6. [保存] を選択します。

スペースの開発環境の停止

開発環境がタイムアウトするように設定されている場合、アイドル状態になる前に実行中の開発環境を停止できます。そうしないと、タイムアウトが経過した開発環境は既に停止されます。開発環境の停止の詳細については、「」を参照してください開発環境の停止。

このページを表示し、スペースレベルで開発環境を管理するには、スペース管理者ロールが必要です。

スペース内の開発環境を停止するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. CodeCatalyst スペースに移動します。

> **Tip**
>
> 複数のスペースに属している場合は、上部のナビゲーションバーでスペースを選択します。

3. 設定 を選択し、開発環境 を選択します。
4. 管理する開発環境の横にあるセレクターを選択します。[Stop] (停止) を選択します。

スペースの開発環境の削除

不要になった開発環境や、所有者がいない開発環境を削除できます。開発環境の削除に関する考慮事項の詳細については、「」を参照してください開発環境の削除。

このページを表示し、スペースレベルで開発環境を管理するには、スペース管理者ロールが必要です。

スペース内の開発環境を削除するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. CodeCatalyst スペースに移動します。

> **Tip**
>
> 複数のスペースに属している場合は、上部のナビゲーションバーでスペースを選択します。

3. 設定 を選択し、開発環境 を選択します。
4. 管理する開発環境の横にあるセレクターを選択します。[削除] を選択します。確認するには、「」と入力しdelete、「削除」を選択します。

スペースのクォータ

次の表に、Amazon のスペースのクォータと制限を示します CodeCatalyst。Amazon のクォータの詳細については CodeCatalyst、「」を参照してくださいのクォータ CodeCatalyst。

スペースの Slack チャネルの最大数	500
E メールアドレスの招待の最大数	25
ユーザーへの招待の最大数	500
あたりのユーザーあたりのアクティブなスペースの最大数 AWS リージョン	5
1 か月、1 リージョン、1 ユーザーあたりのスペース作成の最大数	5
チームの最大 SSO グループ数	5
スペースの最大チーム数	100
チームの最大ユーザー数	1,000
スペースの説明	スペースの説明はオプションです。指定した場合、0～200 文字の長さである必要があります。文字、数字、スペース、ピリオド、アンダースコア、カンマ、ダッシュ、および次の特殊文字の任意の組み合わせを含めることができます。 `? & $ % + = / \ ; : \n \t \r`
スペース名	スペース名は 全体で一意である必要があります CodeCatalyst。削除されたスペースの名前は再利用できません。 スペース名の長さは 3～63 文字にする必要があります。また、英数字で始まる必要があります。スペース名には、文字、数字、ピリオド、アンダースコア、ダッシュを任意に組み合わせ

	て使用できます。次の文字を含めることはできません。 ! ? @ # $ % ^ & * ( ) + = { } [ ] | 八 > < ~ ` ' " ; :

でプロジェクトを操作する CodeCatalyst

Amazon のプロジェクトを使用して CodeCatalyst 、開発チームが共有継続的インテグレーション/継続的デリバリー (CI/CD) ワークフローとリポジトリを使用して開発タスクを実行できるコラボレーションスペースを確立します。プロジェクトを作成するときに、リソースを追加、更新、または削除できます。チームの作業の進行状況をモニタリングすることもできます。1 つのスペースに複数のプロジェクトを含めることができます。

のスペース CodeCatalyst はプロジェクトで構成されます。スペース内のすべてのプロジェクトを表示できますが、使用できるのは自分がメンバーであるプロジェクトのみです。プロジェクトを作成すると、プロジェクトのデフォルトロールが生成され、プロジェクトに招待するユーザーに割り当てることができます。

- Contributor ロール などのプロジェクトロールを持つプロジェクトに割り当てられたユーザーは、ソースリポジトリなどのプロジェクトリソースにアクセスできます。
- Space 管理者プロジェクト管理者またはロールを持つユーザーは、プロジェクトへの参加招待を送信できます。
- プロジェクト管理者ロールを持つユーザーは、共有リソース全体のアクティビティ、ステータス、その他の設定を追跡できます。
- 制限付きアクセスロールを持つユーザーは、CI/CD ワークフローの一部として、機能、コード修正、テストのプロジェクト割り当てを管理できます。

  ワークフローは、CI/CD パイプラインとしてアプリケーションを構築、テスト、リリース、または更新するために使用されます。ソースアーティファクトを転送して処理するアクションを追加することで、ワークフローをアセンブルできます。アクションを実行すると、プロジェクトクラウドリソースを使用してワークフローアクションのオンデマンドコンピューティング機能が提供されます。設定するアクティビティと出力に基づいて、より多くの CI/CD ワークフローを設定できます。例えば、ビルドアクションとテストアクションのみのワークフローを作成すると、バグの修正中にテスト結果を表示し、デプロイなしでワークフローを完了できます。次に、別のワークフローを作成して、アプリケーションをビルドしてステージング環境にデプロイできます。

プロジェクトを作成するときは、設計図を使用して、サンプルコードを含むプロジェクトを作成し、リソースを作成するか、空のプロジェクトから開始できます。設計図を使用してプロジェクトを作成する場合、選択した設計図によって、プロジェクトに追加されるリソースと、プロジェクトリソースを追跡して使用できるように が CodeCatalyst 作成または設定するツールが決まります。プロジェクトを作成した後、リソースを手動で追加または削除できます。

各プロジェクトは、プロジェクトの作成時やリソースの変更時など、ユーザーごとのイベントのリストとしてプロジェクトアクティビティを追跡します。プロジェクトアクティビティは、スペースレベルでモニタリングおよび集計されます。アクティビティデータの使用の詳細については、「」を参照してくださいスペース内のすべてのプロジェクトを表示する。

プロジェクトで AWS リソースを使用している場合は、 CodeCatalyst アカウントを、 AWS プロジェクトのリソースを統合するための管理アクセス許可を持つ アカウントに接続できます。

ソースリポジトリ、問題、その他のリソースは、作成後にプロジェクトに追加できます。プロジェクトを作成するには、スペース管理者ロールが必要です。

「プロジェクトの作成」

CodeCatalyst プロジェクトを使用すると、継続的インテグレーション/継続的デリバリー (CI/CD) ワークフローとリポジトリの共有による開発タスクの実行、リソースの管理、問題の追跡、ユーザーの追加を行うことができます。

プロジェクトを作成する前に、スペース管理者またはパワーユーザーロールが必要です。

トピック

- 設計図を使用したプロジェクトの作成
- Amazon での空のプロジェクトの作成 CodeCatalyst
- リンクされたサードパーティーリポジトリを使用したプロジェクトの作成
- 作成されたプロジェクトへのリソースとタスクの追加

設計図を使用したプロジェクトの作成

プロジェクトのブループリントを使用して、すべてのプロジェクトリソースとサンプルコードをプロビジョニングできます。ブループリントの詳細については、「」を参照してください CodeCatalyst ブループリントを使用した包括的なプロジェクトの作成。

ブループリントを使用してプロジェクトを作成するには

1. CodeCatalyst コンソールで、プロジェクトを作成するスペースに移動します。
2. スペースダッシュボードで、[プロジェクトの作成] を選択します。
3. 設計図 で開始 を選択します。

4. CodeCatalyst ブループリントまたはスペースブループリントタブからブループリントを選択し、次へ を選択します。
5. 「プロジェクトの名前」に、プロジェクトに割り当てる名前とそれに関連するリソース名を入力します。名前はスペース内で一意でなければなりません。
6. (オプション) デフォルトでは、設計図によって作成されたソースコードはリポジトリに CodeCatalyst 保存されます。または、ブループリントのソースコードをサードパーティーのリポジトリに保存することもできます。詳細については、「で拡張機能を使用してプロジェクトに機能を追加する CodeCatalyst」を参照してください。

   使用するサードパーティーのリポジトリプロバイダーに応じて、次のいずれかを実行します。

   - GitHub リポジトリ： GitHub アカウントを接続します。

     詳細ドロップダウンメニューを選択し、リポジトリプロバイダーとして を選択し GitHub、設計図によって作成されたソースコードを保存する GitHub アカウントを選択します。

     > **Note**
     >
     > GitHub アカウントを接続する場合は、アイデンティティと CodeCatalyst アイデンティティの間にアイデンティティマッピングを確立するための個人用接続を作成する必要があります GitHub 。詳細については、「個人用接続」および「個人接続による GitHub リソースへのアクセス」を参照してください。

   - Bitbucket リポジトリ : Bitbucket ワークスペースを接続します。

     詳細ドロップダウンメニューを選択し、リポジトリプロバイダーとして Bitbucket を選択し、設計図によって作成されたソースコードを保存する Bitbucket ワークスペースを選択します。
7. プロジェクトリソース で、設計図パラメータを設定します。設計図によっては、ソースリポジトリ名に名前を付けるオプションがある場合があります。
8. (オプション) 行ったプロジェクトパラメータの選択に基づいて更新を含む定義ファイルを表示するには、「プロジェクトプレビューの生成」から「コードを表示」または「ワークフローを表示」を選択します。
9. (オプション) 設計図のカードから詳細を表示 を選択すると、設計図のアーキテクチャの概要、必要な接続とアクセス許可、設計図が作成するリソースの種類など、設計図に関する特定の詳細が表示されます。
10. [プロジェクトを作成] を選択します。

Amazon での空のプロジェクトの作成 CodeCatalyst

リソースなしで空のプロジェクトを作成し、後で必要なリソースを手動で追加できます。

プロジェクトを作成する前に、スペース管理者またはパワーユーザーロールが必要です。

空のプロジェクトを作成するには

1. プロジェクトを作成するスペースに移動します。
2. スペースダッシュボードで、[プロジェクトの作成] を選択します。
3. [最初から開始] を選択します。
4. [プロジェクトに名前を付ける] に、プロジェクトに割り当てる名前を入力します。名前はスペース内で一意でなければなりません。
5. [プロジェクトを作成] を選択します。

リンクされたサードパーティーリポジトリを使用したプロジェクトの作成

プロジェクトのソースコードを優先サードパーティープロバイダーに保持し、ブループリント、ライフサイクル管理、ワークフローなど、すべての CodeCatalyst 機能を使用できます。これを行うには、 GitHub または Bitbucket リポジトリにリンクする新しい CodeCatalyst プロジェクトを作成できます。その後、リンクされたソースリポジトリを CodeCatalyst プロジェクトで使用できます。

CodeCatalyst プロジェクトを作成する前に、スペース管理者またはパワーユーザーロールが必要です。詳細については、「スペースの作成」および「ユーザーをスペースに直接招待する 」を参照してください。

GitHub アカウントまたは Bitbucket ワークスペースのソースリポジトリにリンク CodeCatalyst するプロジェクトを で作成するには、次の 3 つのタスクを完了する必要があります。

1. GitHub リポジトリまたは Bitbucket 拡張機能をインストールします。外部サイトに接続してサードパーティーリポジトリ CodeCatalyst へのアクセスを提供するように求められます。これは次のステップの一部として行われます。

> **Important**
>
> GitHub リポジトリまたは Bitbucket リポジトリ拡張を CodeCatalyst スペースにインストールするには、スペース管理者ロールを持つアカウントでサインインする必要があります。

2. GitHub アカウントまたは Bitbucket ワークスペースを に接続します CodeCatalyst。

> **Important**
>
> GitHub アカウントまたは Bitbucket ワークスペースを CodeCatalyst スペースに接続するには、サードパーティーのソースの管理者と CodeCatalyst スペース管理者の両方である必要があります。

> **Important**
>
> リポジトリ拡張をインストールすると、リンク先のリポジトリのコード CodeCatalyst のインデックスが作成され、 に保存されます CodeCatalyst。これにより、コードは で検索できるようになります CodeCatalyst。でリンクされたリポジトリを使用する場合のコードのデータ保護について理解を深めるには CodeCatalyst、「Amazon CodeCatalyst ユーザーガイド」の「データ保護」を参照してください。

> **Note**
>
> GitHub アカウントへの接続を使用している場合は、アイデンティティと CodeCatalyst アイデンティティ間のアイデンティティマッピングを確立するために、個人接続を作成する必要があります GitHub 。詳細については、「個人用接続」および「個人接続による GitHub リソースへのアクセス」を参照してください。

3. GitHub リポジトリまたは Bitbucket リポジトリにリンクされた CodeCatalyst プロジェクトを作成します。

> **Important**
>
> GitHub または Bitbucket リポジトリを寄稿者 としてリンクすることはできますが、サードパーティーリポジトリのリンクを解除できるのは、スペース管理者またはプロジェクト管理者 のみです。詳細については、「での GitHub リポジトリ、Bitbucket リポジトリ、Jira プロジェクトのリンク解除 CodeCatalyst」を参照してください。

> **Note**
>
> - は、スペース内の GitHub 1 つの CodeCatalyst プロジェクトにのみリンクできます。
> - 空の GitHub リポジトリまたはアーカイブされたリポジトリを CodeCatalyst プロジェクトで使用することはできません。
> - プロジェクト内の CodeCatalyst リポジトリと同じ名前の GitHub リポジトリをリンクすることはできません。
> - GitHub リポジトリ拡張機能は GitHub Enterprise Server リポジトリと互換性がありません。

詳細については、「で拡張機能を使用してプロジェクトに機能を追加する CodeCatalyst」を参照してください。

サードパーティー拡張機能をインストールするには

1. プロジェクトを作成するスペースに移動します。
2. スペースダッシュボードで、[プロジェクトの作成] を選択します。
3. 「独自のコードを使用」を選択します。
4. GitHub 「既存のリポジトリをリンクする」で、使用するサードパーティーのリポジトリプロバイダーに応じてリポジトリまたは Bitbucket リポジトリを選択します。 GitHub アカウントまたは Bitbucket ワークスペースを接続するように求められます。サードパーティーの拡張機能がまだインストールされていない場合は、インストールプロンプトが表示されます。
5. プロンプトが表示されたら、インストール を選択します。拡張機能に必要なアクセス許可を確認し、続行する場合は、[インストール] を再度選択します。

サードパーティーの拡張機能をインストールした後、次のステップは GitHub アカウントまたは Bitbucket ワークスペースを CodeCatalyst スペースに接続することです。

GitHub または Bitbucket リポジトリプロバイダーを に接続するには CodeCatalyst

設定を選択したサードパーティーの拡張機能に応じて、次のいずれかを実行します。

- GitHub リポジトリ： GitHub アカウントに接続します。

  1. Connect GitHub account を選択して、 の外部サイトに移動します GitHub。

2. GitHub 認証情報を使用して GitHub アカウントにサインインし、Amazon をインストールするアカウントを選択します CodeCatalyst。

   > **Tip**
   >
   > 以前に GitHub アカウントをスペースに接続したことがある場合は、再承認を求められません。代わりに、複数の GitHub スペースのメンバーまたは共同作業者の場合は拡張機能をインストールする場所を尋ねるダイアログボックスが表示され、1 つの GitHub スペースにのみ属している場合は Amazon CodeCatalyst アプリケーションの設定ページが表示されます。許可するリポジトリアクセス用にアプリケーションを設定し、保存 を選択します。保存ボタンがアクティブでない場合は、設定を変更してから再試行してください。

3. CodeCatalyst が現在および将来のすべてのリポジトリにアクセスできるようにするか、 で使用する特定の GitHub リポジトリを選択するかを選択します CodeCatalyst。デフォルトのオプションは、 によってアクセスされる将来の GitHub リポジトリを含め、 GitHub アカウント内のすべてのリポジトリを含めることです CodeCatalyst。
4. に付与されたアクセス許可を確認し CodeCatalyst、インストール を選択します。

GitHub アカウントを に接続すると CodeCatalyst、GitHub リポジトリ拡張の詳細ページが表示されます。このページでは、接続された GitHub アカウントとリンクされた GitHub リポジトリを表示および管理できます。

- Bitbucket リポジトリ : Bitbucket ワークスペースに接続します。

  1. Bitbucket ワークスペースの接続を選択して、Bitbucket の外部サイトに移動します。
  2. Bitbucket 認証情報を使用して Bitbucket ワークスペースにサインインします。
  3. ワークスペースの承認ドロップダウンメニューから、 CodeCatalyst アクセス権を付与する Bitbucket ワークスペースを選択し、アクセス権の付与を選択します。

     > **Tip**
     >
     > 以前に Bitbucket ワークスペースをスペースに接続したことがある場合は、再承認を求められません。代わりに、複数の Bitbucket ワークスペースのメンバーまたは共同作業者の場合は拡張機能をインストールする場所を尋ねるダイアログが表示され、1 つの Bitbucket ワークスペースにのみ属している場合は Amazon CodeCatalyst アプリケーションの設定ページが表示されます。許可するワークスペースアクセス用にアプ

リケーションを設定し、アクセス許可 を選択します。アクセス許可ボタンがアクティブでない場合は、設定を変更してから、もう一度試してください。

Bitbucket ワークスペースを に接続すると CodeCatalyst、Bitbucket リポジトリ拡張の詳細ページが表示されます。このページでは、接続されている Bitbucket ワークスペースとリンクされた Bitbucket リポジトリを表示および管理できます。

サードパーティーのリポジトリプロバイダーを に接続したら CodeCatalyst、サードパーティーのリポジトリを CodeCatalyst プロジェクトにリンクできます。

プロジェクトを作成するには

1. プロジェクトの作成ページで、接続した GitHub アカウントまたは Bitbucket ワークスペースを選択します。
2. 接続したサードパーティーのリポジトリプロバイダーに応じて、GitHub リポジトリまたは Bitbucket リポジトリのドロップダウンメニューを選択してサードパーティーのリポジトリを表示し、プロジェクトにリンクするリポジトリを選択します。
3. 「プロジェクトテキスト入力の名前」フィールドに、プロジェクトに割り当てる名前を入力します。名前はスペース内で一意でなければなりません。
4. [プロジェクトを作成] を選択します。

GitHub リポジトリまたは Bitbucket リポジトリ拡張機能をインストールし、リソースプロバイダーを接続し、サードパーティーのリポジトリを CodeCatalyst プロジェクトにリンクしたら、CodeCatalyst ワークフローや開発環境で使用できます。ブループリントから生成されたコードを使用して、接続された GitHub アカウントまたは Bitbucket ワークスペースにサードパーティーのリポジトリを作成することもできます。詳細については、「サードパーティーのリポジトリイベント後にワークフローを自動的に開始する」および「開発環境の作成」を参照してください。

作成されたプロジェクトへのリソースとタスクの追加

プロジェクトの準備ができたら、リソースとタスクを追加できます。

- プロジェクトで作成された CI/CD ワークフローの詳細については、「」を参照してくださいワークフローの開始方法。

- ビルドアーティファクトを Amazon S3 バケットにデプロイする新しいプロジェクトのビルドアクションと同様のビルドアクションを操作するには、ワークフローによる構築「」および「」を参照してくださいチュートリアル: Amazon S3 にアーティファクトをアップロードする。
- 空のプロジェクトから開始し、 AWS CloudFormation スタックデプロイで同様のサーバーレスアプリケーションのデプロイを操作するには、「」を参照してくださいチュートリアル: を使用してサーバーレスアプリケーションをデプロイする AWS CloudFormation。
- 問題計画ボードを追加するには、「」を参照してくださいで問題のある作業を追跡して整理する CodeCatalyst。
- プロジェクトの概要、プロジェクトのステータス、最近のチームアクティビティ、割り当てられた作業を確認するには、「」を参照してくださいプロジェクトのリストの取得。
- ソースコードを表示したり、プルリクエストを作成したりするには、「」を参照してくださいでソースリポジトリを使用してコードを保存し、共同作業する CodeCatalyst。
- ワークフロー実行の成功または失敗のステータスアラートを送信する通知を設定するには、「」を参照してくださいAmazon での通知の管理 CodeCatalyst。
- メンバーをプロジェクトに招待するには、「」を参照してくださいユーザープロジェクトのアクセス許可の付与。
- 開発環境を設定するには、「」を参照してくださいで開発環境を使用してコードを記述および変更する CodeCatalyst。

プロジェクトのリストの取得

CodeCatalyst スペースから、プロジェクトアクセス許可がある各プロジェクトの詳細を表示できます。

プロジェクトを表示するには、プロジェクトのメンバーであるか、スペースのスペース管理者ロールを持っている必要があります。

プロジェクトをまだ作成していない場合は、「」を参照してください「プロジェクトの作成」。プロジェクトを作成するスペースには、スペース管理者ロールが必要です。

- プロジェクトの概要では、プロジェクトメンバー、ソースリポジトリ、ワークフロー実行、オープンプルリクエスト、プロジェクト開発環境、および問題を表示できます。
- プロジェクト設定では、プロジェクトの詳細の表示と管理、プロジェクトの削除、プロジェクトへの新規メンバーの招待、プロジェクトメンバーの管理、通知の設定を行うことができます。

プロジェクトタスクと開発環境の表示

未解決の問題やプルリクエストなど、ユーザーに割り当てられたか、ユーザーによって作成されたプロジェクトタスク、およびプロジェクトに関連付けられた開発環境の概要を表示するには、 コンソールを使用します。

プロジェクトを表示するには、プロジェクトのメンバーであるか、スペースのスペース管理者ロールを持っている必要があります。

ソースリポジトリ、ワークフロー実行、問題、プルリクエスト、開発環境、および問題を表示するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 表示するプロジェクトがあるスペースに移動します。プロジェクト で、プロジェクトを選択します。
3. ナビゲーションペインで、[Overview (概要)] を選択します。
4. 自分に割り当てられたプロジェクトタスクと、自分によって作成されたプロジェクトタスクを表示します。

   - メンバーを表示する + すべてのリストを表示して、プロジェクトメンバーのリストを表示します。
   - リポジトリカードを表示して、プロジェクトに関連付けられているソースリポジトリを表示します。
   - ワークフロー実行カードを表示して、プロジェクトに関連付けられているワークフローを表示します。
   - Open pull requests card を表示して、ユーザーに割り当てられたプルリクエストとユーザーによって作成されたプルリクエストに加えて、コードリポジトリのステータスの概要を表示します。
   - マイ開発環境カードを表示して、プロジェクトに関連付けられている開発環境の概要を表示します。
   - 問題カードを表示して、割り当てられたタスクまたは作成したタスクの概要を表示します。

スペース内のすべてのプロジェクトを表示する

スペースのプロジェクトリストで、アクセス許可を持つすべてのプロジェクトを表示できます。

未解決の問題やプルリクエストなど、自分に割り当てられたか、自分によって作成されたプロジェクトタスク、およびプロジェクトに関連付けられた開発環境の概要を表示するには、 コンソールを使用します。

プロジェクトを表示するには、プロジェクトのメンバーであるか、スペースのスペース管理者ロールを持っている必要があります。

ソースリポジトリ、ワークフロー実行、問題、プルリクエスト、開発環境、および問題を表示するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 表示するプロジェクトがあるスペースに移動します。プロジェクト で、プロジェクトを選択します。
3. ナビゲーションペインで、プロジェクト設定 を選択します。
4. プロジェクト名、パス、プロジェクト ID、説明を表示します。

プロジェクト設定の表示

プロジェクト設定 では、プロジェクトメンバー、ソースリポジトリ、ワークフロー実行、オープンプルリクエスト、プロジェクト開発環境、および問題を表示できます。

未解決の問題やプルリクエストなど、ユーザーに割り当てられたか、ユーザーによって作成されたプロジェクトタスク、およびプロジェクトに関連付けられた開発環境の概要を表示するには、 コンソールを使用します。

ソースリポジトリ、ワークフロー実行、問題、プルリクエスト、開発環境、および問題を表示するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 表示するプロジェクトがあるスペースに移動します。プロジェクト で、プロジェクトを選択します。
3. ナビゲーションペインで、プロジェクト設定 を選択します。
4. プロジェクト名、パス、プロジェクト ID、説明を表示します。

で別のプロジェクトに変更する CodeCatalyst

別のプロジェクトに変更するには、コンソールを使用して、アクセスできるプロジェクトのリストから選択します。

別のプロジェクトに変更するには

1. CodeCatalyst コンソールで、上部にあるプロジェクトセレクタを選択します。
2. ドロップダウンを展開し、移動するプロジェクトを選択します。

プロジェクトの削除

プロジェクトを削除して、プロジェクトのリソースへのすべてのアクセスを削除できます。プロジェクトを削除するには、スペース管理者またはプロジェクト管理者ロールが必要です。プロジェクトを削除すると、プロジェクトメンバーはプロジェクトリソースにアクセスできなくなり、サードパーティーのソースリポジトリによって求められるワークフローはすべて停止します。

プロジェクトを削除するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 表示するプロジェクトがあるスペースに移動します。プロジェクト で、プロジェクトを選択します。
3. ナビゲーションペインで、プロジェクト設定 を選択します。
4. [プロジェクトを削除] を選択します。
5. **`delete`** を入力して削除を確定します。
6. [プロジェクトを削除] を選択します。

ユーザープロジェクトのアクセス許可の付与

Amazon CodeCatalyst コンソールを使用して、プロジェクトのメンバーを管理できます。ユーザーの追加や削除、現在のメンバーのロールの管理、プロジェクトへの招待の送信、まだ承諾されていない招待のキャンセルを行うことができます。

スペースユーザーとプロジェクトユーザーのメンバーページで、ユーザーは複数のロールを持つことができます。複数のロールを持つユーザーには、複数のロールがある場合にインジケータが表示され、最もアクセス許可の高いロールが最初に表示されます。

メンバーとそのプロジェクトロールのリストの取得

プロジェクトにユーザーを追加すると、次のようにプロジェクトのアクセス許可を付与するロールが割り当てられます。

- プロジェクト管理者ロールには、プロジェクト内のすべてのアクセス許可があります。このロールは、プロジェクト設定の編集、プロジェクトのアクセス許可の管理、プロジェクトの削除など、プロジェクトのあらゆる側面を管理する必要があるユーザーにのみ割り当てます。詳細については、「プロジェクト管理者ロール」を参照してください。
- Contributor ロールには、プロジェクトで作業するために必要なアクセス許可があります。プロジェクトでコード、ワークフロー、問題、アクションを操作する必要があるユーザーにこのロールを割り当てます。詳細については、「寄稿者ロール」を参照してください。
- レビューワーロールにはレビュー権限があります。詳細については、「レビューワーロール」を参照してください。
- 読み取り専用ロールには読み取りアクセス許可があります。詳細については、「読み取り専用ロール」を参照してください。

Space 管理者ロールを持つユーザーをプロジェクトに招待する必要はありません。ユーザーは、スペース内のすべてのプロジェクトに既に暗黙的にアクセスできるためです。

プロジェクトにユーザーを招待すると (スペース管理者ロールを割り当てずに)、ユーザーはプロジェクトメンバーテーブルの下のプロジェクト、およびスペースの下のプロジェクトメンバーテーブルに表示されます。

スペース内のユーザーとロールを表示するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 表示するプロジェクトがあるスペースに移動します。プロジェクト で、プロジェクトを選択します。
3. ナビゲーションペインで、プロジェクト設定 を選択します。
4. [メンバー] タブを選択します。

   プロジェクトメンバーテーブルには、プロジェクト内のロールを持つすべてのメンバーが表示されます。

> **Tip**
>
> Space 管理者ロールがある場合は、直接招待されたプロジェクトを表示できます。プロジェクトのプロジェクト設定に移動し、マイプロジェクト を選択します。

Space 管理者テーブルには、Space 管理者ロールを持つユーザーが表示されます。これらのユーザーは、スペース内のすべてのプロジェクトに自動的に (暗黙的に) 割り当てられ、プロジェクトにロールはありません。

Status 列では、有効な値は次のとおりです。

- 招待済み – 招待 CodeCatalyst を送信しましたが、ユーザーはまだ承諾または拒否していません。
- メンバー – ユーザーは招待を受け入れました。

トピック

プロジェクトへのユーザーの招待

コンソールを使用して、プロジェクトにユーザーを招待できます。スペースのメンバーを招待したり、スペースの外部から名前を追加したりできます。

プロジェクトにユーザーを招待するには、プロジェクト管理者またはスペース管理者ロールでサインインする必要があります。

Space 管理者ロールを持つユーザーをプロジェクトに招待する必要はありません。ユーザーは、スペース内のすべてのプロジェクトに既に暗黙的にアクセスできるためです。

プロジェクトにユーザーを招待すると (スペース管理者ロールを割り当てずに)、ユーザーはプロジェクトメンバーテーブルの下のプロジェクト、およびスペースの下のプロジェクトメンバーテーブルに表示されます。

プロジェクト設定タブからメンバーをプロジェクトに招待するには

1. プロジェクトに移動します。

   > **Tip**
   >
   > 上部のナビゲーションバーに表示するプロジェクトを選択できます。

2. ナビゲーションペインで、プロジェクト設定 を選択します。
3. [メンバー] タブを選択します。
4. プロジェクトメンバー で、新しいメンバーの招待 を選択します。
5. 新しいメンバーの E メールアドレスを入力し、このメンバーのロールを選択し、招待を選択します。ロールの詳細については、「ユーザーロールによるアクセス許可の付与」をご参照ください。

プロジェクトの概要ページからプロジェクトにメンバーを招待するには

1. プロジェクトに移動します。

   > **Tip**
   >
   > 上部のナビゲーションバーに表示するプロジェクトを選択できます。

2. メンバー + ボタンを選択します。
3. 新しいメンバーの E メールアドレスを入力し、このメンバーのロールを選択し、招待 を選択します。ロールの詳細については、「ユーザーロールによるアクセス許可の付与」をご参照ください。

招待のキャンセル

最近招待を送信した場合は、招待がまだ承諾されていない限りキャンセルできます。

プロジェクトの招待を管理するには、プロジェクト管理者またはスペース管理者ロールが必要です。

プロジェクトメンバーの招待をキャンセルするには

1. キャンセルする招待を送信したプロジェクトに移動します。
2. ナビゲーションペインで、プロジェクト設定 を選択します。

3. メンバータブを表示し、メンバーのステータスが招待された であることを確認します。

> **Note**
>
> キャンセルできるのは、まだ承諾されていない招待のみです。

4. 招待されたメンバーを含む行の横にあるオプションを選択し、招待をキャンセル を選択します。
5. 確認ウィンドウが表示されます。「招待をキャンセルする」を選択して確認します。

プロジェクトからのユーザーの削除

コンソールを使用して、プロジェクトからユーザーを削除できます。

プロジェクトからユーザーを削除するには、プロジェクト管理者またはスペース管理者ロールでサインインする必要があります。

> **Note**
>
> スペース内のすべてのプロジェクトからユーザーを削除すると、そのスペースからユーザーが自動的に削除されます。

プロジェクトからユーザーを削除するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 表示するプロジェクトがあるスペースに移動します。プロジェクト で、プロジェクトを選択します。
3. ナビゲーションペインで、プロジェクト設定 を選択します。
4. [メンバー] タブを選択します。
5. 削除するプロファイルの横にあるセレクターを選択し、 の削除を選択します。
6. ユーザーを削除することを確認し、削除を選択します。

プロジェクトへの招待の承諾または拒否

Amazon CodeCatalyst プロジェクトに参加するための招待メールが届く場合があります。招待を承認または拒否することができます。

招待を承諾または辞退するには

1. 招待メールを開きます。
2. E メール内のプロジェクトリンクを選択します。
3. 「承諾」または「拒否」を選択します。

   拒否 を選択すると、招待を拒否したことを通知するメールがプロジェクト管理アカウントに送信されます。

チームを使用したプロジェクトアクセスの許可

プロジェクトを作成したら、チームを追加できます。チームを使用すると、ユーザーをグループ化して、アクセス許可を共有し、プロジェクトやスペースのメンバー CodeCatalyst として 内のプロジェクト、問題の追跡、ロール、リソースを管理できます。

プロジェクトのチームを管理するには、プロジェクト管理者ロールが必要です。

チームも のスペースレベルで管理されます CodeCatalyst。スペース内のチームの詳細については、「」を参照してくださいチームを使用したスペースアクセスの許可。

トピック

- プロジェクトへのチームの追加
- チームへのプロジェクトロールの付与
- チームのプロジェクトロールの削除

プロジェクトへのチームの追加

チームメンバーがプロジェクトのリソースにアクセスできるチームを管理できます。

スペースユーザーとプロジェクトユーザーのメンバーページで、ユーザーは複数のロールを持つことができます。複数のロールを持つユーザーには、複数のロールがある場合にインジケータが表示され、最もアクセス許可の大きいロールが最初に表示されます。

チームを追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトに移動します。プロジェクト設定 を選択し、チーム を選択します。

3. チームの追加 を選択します。
4. チーム で、利用可能なチームのリストからチームを選択します。
5. プロジェクトロール で、 で使用可能なプロジェクトロールのリストからロールを選択します CodeCatalyst。
   - プロジェクト管理者 — 詳細については、「」を参照してくださいプロジェクト管理者ロール。
   - Contributor — 詳細については、「」を参照してください寄稿者ロール。
   - レビューワー — 詳細については、「」を参照してくださいレビューワーロール。
   - 読み取り専用 — 詳細については、「」を参照してください読み取り専用ロール。
6. チームの追加 を選択します。

チームへのプロジェクトロールの付与

チームには、スペース内の Power ユーザー などのロール許可があります。チームのスペースロールは変更できますが、チームのすべてのメンバーがこれらのアクセス許可を継承することに注意してください。

プロジェクトロールを追加または変更するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 自分のスペースに移動します。プロジェクト設定 を選択し、チーム を選択します。
3. ロールを変更するには、このリストのチームの横にあるセレクターを選択し、ロールの変更を選択します。ロールを追加するには、「プロジェクトロールの追加」を選択します。プロジェクト で追加するプロジェクトを選択し、ロール でロールを選択します。使用可能なプロジェクトロールのいずれかを選択します。
   - プロジェクト管理者 - 詳細については、「」を参照してくださいプロジェクト管理者ロール。
   - Contributor - 詳細については、「」を参照してください寄稿者ロール。
   - レビューワー - 詳細については、「」を参照してくださいレビューワーロール。
   - 読み取り専用 - 詳細については、「」を参照してください読み取り専用ロール。
4. [保存] を選択します。

チームのプロジェクトロールの削除

では CodeCatalyst、チームのプロジェクトロールを表示できます。チームのメンバーを表示することもできます。チームのプロジェクトロールを削除できます。

プロジェクトロールを削除するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 自分のスペースに移動します。プロジェクト設定 を選択し、チーム を選択します。
3. プロジェクトロールタブを選択します。
4. 削除するロールを選択します。

> **Important**
>
> チームからロールを削除すると、チーム内のすべてのユーザーに関連付けられたアクセス許可が削除されます。

5. [保存] を選択します。

マシンリソースへのプロジェクトアクセスを許可する

マシンリソースは、 内のプロジェクトまたはスペースに対するアクセス許可 CodeCatalystthat が付与される 内の特定のリソースです CodeCatalyst。

> **Note**
>
> マシンリソースという用語は、EC2 インスタンスなどのクラウドインフラストラクチャを指すのではなく、スペースまたはプロジェクトのアクセス許可を持つ設計図またはワークフローリソースを指すためのものです。

プロジェクトでマシンリソースを操作する例としては、ブループリントリソースがユーザーに代わってプロジェクトにアクセスできるようにするなどがあります。

マシンリソースは、 SSO CodeCatalyst 経由で にアクセスするときに、承認されたリソースからの ID を表します。マシンリソースは、設計図やワークフロー などのプロジェクト内のリソースにアクセス許可を付与するために使用されます。プロジェクト内のマシンリソースを表示したり、プロジェ

クトのマシンリソースを有効または無効にしたりできます。例えば、マシンリソースを無効にしてアクセスを管理し、後で再度有効にすることができます。

これらのオペレーションは、マシンリソースを取り消したり無効にしたりする必要がある場合に、マシンリソースで使用できます。例えば、認証情報が侵害された可能性がある場合は、マシンリソースを無効にできます。通常、これらのオペレーションを使用する必要はありません。

このページを表示し、プロジェクトレベルでマシンリソースを管理するには、スペース管理者ロールまたはプロジェクト管理者ロールが必要です。

マシンリソースは、 のスペースレベルで管理されます CodeCatalyst。スペース/プロジェクトのチームの詳細については、「」を参照してくださいマシンリソースのスペースアクセスを許可する 。

トピック

- マシンリソースのプロジェクトアクセスの表示
- マシンリソースのプロジェクトアクセスを無効にする
- マシンリソースのプロジェクトアクセスの有効化

マシンリソースのプロジェクトアクセスの表示

プロジェクトで使用されているマシンリソースのリストを表示できます。

Space 管理者ロールまたはプロジェクト管理者ロールが必要です。

マシンリソースを表示するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトに移動し、プロジェクト設定 を選択します。マシンリソース を選択します。
3. ドロップダウンで、ワークフローアクションを選択して、ワークフローのマシンリソースのみを表示します。ブループリントを選択すると、ブループリントのマシンリソースのみが表示されます。

   フィルターフィールドを使用して名前をフィルタリングすることもできます。

マシンリソースのプロジェクトアクセスを無効にする

プロジェクトで使用されているマシンリソースを無効にすることを選択できます。

> **Important**
>
> マシンリソースを無効にすると、スペース内のすべての関連するブループリントまたはワークフローに対するすべてのアクセス許可が削除されます。

Space 管理者ロールまたはプロジェクト管理者ロールが必要です。

マシンリソースを無効にするには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトに移動し、プロジェクト設定 を選択します。マシンリソース を選択します。
3. 次のいずれかを選択します 。

> **Important**
>
> マシンリソースを無効にすると、スペース内のすべての関連するブループリントまたはワークフローに対するすべてのアクセス許可が削除されます。

- 個別に無効にするには、無効にする 1 つ以上のマシンリソースの横にあるセレクターを選択します。無効化 を選択し、このリソース を選択します。
- すべてのリソースを無効にするには、 を無効化 を選択し、すべてのリソース を選択します。
- すべてのワークフローアクションを無効にするには、 を無効にする を選択し、次にすべてのワークフローアクション を選択します。
- すべてのブループリントを無効にするには、 を無効化 を選択し、すべてのブループリント を選択します。

マシンリソースのプロジェクトアクセスの有効化

プロジェクトで使用中で無効になっているマシンリソースを有効にすることができます。

Space 管理者ロールまたはプロジェクト管理者ロールが必要です。

マシンリソースを有効にするには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。

2. プロジェクトに移動し、プロジェクト設定 を選択します。マシンリソース を選択します。
3. 次のいずれかを選択します 。

   - を個別に有効にするには、有効にする 1 つ以上のマシンリソースの横にあるセレクターを選択します。を有効化 を選択し、このリソース を選択します。
   - すべてのリソースを有効にするには、「 を有効にする」を選択し、「すべてのリソース」を選択します。
   - すべてのワークフローアクションを有効にするには、「 を有効にする」を選択し、「すべてのワークフローアクション」を選択します。
   - すべてのブループリントを有効にするには、 を有効にする を選択し、すべてのブループリント を選択します。

プロジェクトのクォータ

次の表に、Amazon のプロジェクトのクォータと制限を示します CodeCatalyst。Amazon のクォータの詳細については CodeCatalyst、「」を参照してくださいのクォータ CodeCatalyst。

スペースあたりのプロジェクトの最大数	100
ユーザーが属できるプロジェクトの最大数	1,000
プロジェクトに属せるメンバーの最大数。	10,000
プロジェクト名	プロジェクト名はスペース内で一意である必要があります。名前は 3～63 文字にする必要があります。名前では、大文字と小文字が区別されます。プロジェクト名は英数字で始める必要があります。有効な文字: A～Z、a～z、0～9、スペース、および 。、_ (アンダースコア) - (ハイフン) プロジェクト名には、次の文字を含めることはできません。 ! ? @ # $ % ^ & * ( ) + = { } [ ] | \ / > < ~ ` ' " ; :
プロジェクトの説明	プロジェクトの説明は最大 200 文字です。有効な文字: A～Z、a～z、0～9、スペース、.、_

(アンダースコア) - (ハイフン) 。プロジェクトの説明はオプションです。

での通知の使用 CodeCatalyst

でプロジェクトとリソースをモニタリングする通知を設定できます CodeCatalyst。ユーザーは、自分がメンバーである任意のプロジェクトで E メールを受信するプロジェクトイベントを選択できます。また、 CodeCatalyst スペースと Slack ワークスペース間のアクセスを設定し、その Slack ワークスペース内の 1 つ以上のチャネルにプロジェクトを送信するように通知を設定することで、Slack などのチームメッセージングアプリケーション内のチーム全体に送信する通知を設定することもできます。 CodeCatalyst スペースと Slack ワークスペース間のアクセスを設定すると、プロジェクトメンバーは独自の Slack メンバー IDs を追加して、接続された Slack ワークスペースとチャネルの CodeCatalyst イベントについて直接通知されるようにすることもできます。

> **Note**
>
> Slack に送信できるプロジェクトイベントのセットは、ユーザーが E メールで通知することを選択できるイベントセットと同じではありません。

トピック

通知はどのような仕組みで機能しますか?

Slack などのチームメッセージングアプリケーションに通知を送信するようにプロジェクトを設定できます。

通知にはどのような権限が必要ですか?

プロジェクトメンバーなら誰でも、チャンネルの通知設定を設定、表示、更新、削除できます CodeCatalyst。ただし、Slack ワークスペースを追加または削除できるのはスペース管理者権限を持つユーザーだけです。すべてのユーザーは、自分が所属するプロジェクトについて、どのプロジェクトイベントに関するメールを受信したいかを設定できます。 CodeCatalyst

CodeCatalyst どのイベントに関する通知を設定できますか?

CodeCatalyst ワークフローイベントに関する通知を 1 つ以上の Slack チャンネルに配信するように設定できます。 CodeCatalyst プロジェクトと Slack の間で通知を設定すると、プロジェクトユーザーは自分の Slack メンバー ID を追加して、Slack チャンネルでイベントに関するダイレクトメッセージを受信できるようになります。 CodeCatalyst Slack メンバー ID を追加したユーザーは、プロジェクト用に設定されている Slack チャンネルで自分の ID へのダイレクトメンションを受け取るので、気になるイベントについての認知度を高めるのに役立ちます。

また、どのイベントについてメールを受信するかも選択できます。これらのメールは、AWS Builder ID に設定されたメールアドレスに送信されます。

通知はどのように表示されますか?

1 つ以上の Slack CodeCatalyst チャンネルに通知を配信するように設定できます。Slack CodeCatalyst ワークスペースにアクセスする権限を付与するには承認が必要です。権限が与えられると、設定した Slack CodeCatalyst チャンネルに通知を配信できます。プロジェクトメンバーが Slack メンバー ID を追加することを選択した場合、そのプロジェクトに設定されている Slack CodeCatalyst チャンネルでイベントに関するメンションを受け取ることができます。

通知を設定するにはどうすればいいですか ?

メール通知はの一部として設定されます CodeCatalyst。プロジェクトユーザーは、メールを受信したいイベントを [マイ設定] ページで選択できます。

プロジェクトリソースに Slack 通知を設定するには、以下の大まかなタスクを完了する必要があります。

通知 (上位タスク) を設定するには

1. では CodeCatalyst、と Slack CodeCatalyst などのメッセージングクライアントとの接続を設定します。Slack ワークスペースが接続されると、そのワークスペースはスペース内のすべてのプロジェクトで利用できるようになります。

   > **Note**
   >
   > Slack ワークスペースを追加または削除できるのは、スペース管理者権限を持つユーザーだけです。

2. のプロジェクトで CodeCatalyst、チームに通知を受け取りたいチャンネルを追加します。

3. では CodeCatalyst、ワークフローの実行失敗など、さまざまなイベントの通知をオンにし、通知を送信したいチャネルを指定します。

詳細なステップについては、「Slack 通知を使い始める」を参照してください。

CodeCatalyst スペースと Slack の間で通知を設定すると、ユーザーは自分の Slack メンバー ID を追加して、プロジェクトに設定された Slack CodeCatalyst チャンネルでイベントに関するダイレクトメッセージを受信できるようになります。

Slack 通知を使い始める

プロジェクトを作成したら、チームがプロジェクトリソースを監視するのに役立つ Slack 通知を設定できます。

以下の手順では、初めて Slack 通知を設定する手順を説明します。 CodeCatalyst通知をすでに設定している場合は、を参照してくださいAmazon での通知の管理 CodeCatalyst。

> **Note**
>
> 通知チャネルに送信できる一連のプロジェクトイベントは、ユーザーが電子メールで通知を受けるように選択できる一連のイベントとは異なります。詳細については、「Amazon での通知の管理 CodeCatalyst」を参照してください。

トピック

前提条件

開始するには、以下が必要です。

- CodeCatalyst スペース。 CodeCatalyst スペースを作成して初めてサインインする方法については、を参照してくださいのセットアップと へのサインイン CodeCatalyst。

- CodeCatalyst プロジェクト。詳細については、「「プロジェクトの作成」」を参照してください。
- CodeCatalyst プロジェクト管理者またはスペース管理者の役割を持つアカウント。詳細については、「ユーザーロールによるアクセス許可の付与」を参照してください。
- からアクセスできる Slack アカウントと Slack ワークスペース。 CodeCatalyst
- 通知を送信する Slack チャンネル CodeCatalyst 。チャンネルは公開でも非公開でもかまいません。

ステップ 1: Slack CodeCatalyst ワークスペースConnect する

スペース管理者権限を持つユーザーのみが Slack ワークスペースを追加または削除できます。Slack ワークスペースを追加または削除すると、スペース内のすべてのプロジェクトに影響します。と Slack との接続を確立するには、Slack CodeCatalyst ワークスペースと安全な OAuth CodeCatalyst 認証ハンドシェイクを実行します。

以下の手順で Slack ワークスペースに接続します CodeCatalyst 。

> **Note**
>
> これは Slack ワークスペースごとに 1 回だけ行う必要があります。その後、Slack チャンネルごとに通知を設定できます。

Slack CodeCatalyst ワークスペースに接続するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. プロジェクトに移動します。
3. ナビゲーションペインで [プロジェクト設定] を選択します。
4. [通知] タブを選択します。
5. [通知の設定] を選択します。
6. 「Slack ワークスペースにConnect」を選択します。
7. ダイアログボックスの内容を読み、「Slack ワークスペースにConnect」を選択します。
8. AWSChatbot メッセージで:
   a. 右上で、チャンネルを含む Slack ワークスペースを選択します。
   b. [Allow] (許可) を選択します。

CodeCatalyst コンソールに戻ります。

9. 「ステップ 2: Slack チャンネルをに追加する CodeCatalyst」に進みます。

ステップ 2: Slack チャンネルをに追加する CodeCatalyst

チャンネルを追加するには Slack チャンネル ID が必要です。 CodeCatalyst

Slack チャンネル ID を取得するには

1. Slack にサインインします。詳しくは、「Slack へのサインイン」を参照してください。
2. 通知を送りたいチャンネルが含まれている Slack ワークスペースに移動します。詳しくは、「Slack ワークスペース間の切り替え」または「他の Slack ワークスペースへのログイン」を参照してください。
3. ナビゲーションペインで、通知を送りたいチャンネルのコンテキスト (右クリック) メニューを開き、「チャンネルの詳細を開く」を選択します。

   チャネル ID はダイアログボックスの下部に表示されます。
4. チャネル ID の値をコピーします。これは次のステップで必要になります。

コピーしたチャンネル ID を使って、Slack CodeCatalyst チャンネルを接続できるようになりました。

Slack チャンネルをに追加するには CodeCatalyst

1. 始める前に、Slack チャンネルがプライベートの場合は、以下のように AWS Chatbot アプリをチャンネルに追加します。

   a. Slack チャンネルのメッセージボックスに「aws app」**@aws** と入力し、ダイアログボックスから「aws app」を選択します。

   b. [Enter] キーを押します。

      AWSチャットボットがプライベートチャンネルにいないことを示す Slackbot メッセージが表示されます。

   c. 「招待する」を選択して、AWS Chatbot チャンネルに招待します。
2. CodeCatalyst コンソールで [次へ] を選択します。

3. 「チャンネル ID」に、先ほど取得した Slack チャンネル ID を貼り付けます。
4. 「チャンネル名」に名前を入力します。Slack のチャンネル名を使用することをお勧めします。
5. [次へ] をクリックします。
6. 「通知イベントの選択」で、通知を受け取りたいイベントの種類を選択します。
7. [終了] を選択します。

ステップ 3: Slack CodeCatalyst への通知をテストする

ワークフローステータスの通知を送信するようにプロジェクトを設定したら、Slack で通知を確認できます。

Slack で通知を表示するには

1. CodeCatalyst プロジェクト内でワークフローを手動で開始すると、ワークフローの実行が完了し、実行が完了するとステータス通知が届きます。
2. Slack で、通知用に設定したチャンネルを表示します。通知には、各ワークフロー実行の最新ステータスと、失敗か成功かが示されます。

ステップ 4: 次のステップ

CodeCatalyst スペースに Slack ワークスペースを設定したら、 CodeCatalyst 既存のプロジェクトに Slack チャンネルを追加したり、作成後に新しいプロジェクトに追加したりできます。また、プロジェクトユーザーに Slack メンバー ID の個人用 Slack 通知を設定できることや、メールを受信するイベントを設定できることを知らせることもできます。詳細については、「Amazon での通知の管理 CodeCatalyst」を参照してください。

Amazon での通知の管理 CodeCatalyst

CodeCatalyst プロジェクト内のイベントに関する通知を送信するように設定できます。Slack チャンネルなどのメッセージングクライアントに通知を送信できます。プロジェクトユーザーは、プロフィールに設定されたメールアドレスに送信されるメールで、どのプロジェクトイベントについて通知を受けるかを選択できます。

> **Note**
>
> 通知チャネルに送信できる一連のプロジェクトイベントは、ユーザーが電子メールで通知を受け取るように選択できる一連のイベントとは異なります。

トピック

自分に直接送信される通知を管理する

メンバーになっているどのプロジェクトでも、イベントに関するメール通知を自分宛に送信するように選択できます。これらのメールは、AWS Builder ID で設定したメールアドレスに送信されます。デフォルトでは、メールを送信できるすべてのプロジェクトイベントに関するメールが届きます。

プロジェクトが Slack チャンネルに通知を送信するように設定されている場合、Slack メンバー ID を追加すると、その Slack CodeCatalyst チャンネルのイベントに関するダイレクトメンションを受け取ることができます。これにより、自分が担当しているプロジェクトで起きているイベントについての認識を高めることができます。

プロジェクトイベントのメール通知を設定するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. 上部のメニューバーでプロファイルバッジを選択し、[My 設定] を選択します。[CodeCatalyst マイセッティング] ページが開きます。

   > **Tip**
   >
   > プロジェクトまたはスペースのメンバーページに移動し、メンバーリストから自分の名前を選択して、ユーザープロファイルを確認することもできます。

3. 「メール通知」で、メール通知を設定したいプロジェクトをリストから探し、「編集」を選択します。
4. メールを受信したいイベントを選択し、[保存] を選択します。

Slack の個人通知を設定するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. 上部のメニューバーでプロファイルバッジを選択し、[My 設定] を選択します。[CodeCatalyst マイセッティング] ページが開きます。

> **Tip**
>
> プロジェクトまたはスペースのメンバーページに移動し、メンバーリストから自分の名前を選択して、ユーザープロファイルを確認することもできます。

3. 「パーソナル Slack 通知」で「Slack ID をConnect」を選択し、「Slack ワークスペースに Connect」を選択します。別のウィンドウが開きます。

> **Tip**
>
> このオプションは、スペース管理者権限を持つユーザーがスペースに Slack ワークスペースを追加しない限り、設定できません。 CodeCatalyst詳細については、「Slack 通知を使い始める」および「チャンネルに送信される通知の管理」を参照してください。

4. 権限リクエストウィンドウで、ワークスペースの名前がスペースに設定されている Slack ワークスペースと一致していることを確認してください。 CodeCatalyst 「許可」AWS Chatbot を選択してワークスペースへのアクセスを許可します。ウィンドウが閉じ、Slack ワークスペースの接続ステータスが「接続済み」と表示されます。

> **Tip**
>
> 接続ステータスが変わらない場合は、Slack ワークスペースへの接続中にエラーが発生していないか確認してください。エラーを確認するには上へのスクロールが必要な場合があります。

5. 個人用 Slack 通知の受信を停止するには、接続している Slack ワークスペースを選択し、「Slack ID を連携解除」を選択します。

チャンネルに送信される通知の管理

チーム Slack CodeCatalyst チャンネルなどのチームリソースに送信されるプロジェクトイベントに関する通知を追加して管理することができます。そうすることで、ワークフローの実行が失敗したときなど、重要なイベントをチーム全体が確実に把握できるようになります。

> **Note**
>
> プロジェクトのメンバーなら誰でも、そのプロジェクトのチャンネルに送信される通知を管理できます。ただし、Slack ワークスペースを追加または削除できるのはスペース管理者権限を持つユーザーだけです。

プロジェクトへの通知チャンネルの追加

チームの Slack チャンネルなど、通知を受け取りたいチャンネルを追加できます。

通知用の Slack チャンネルを追加するには

1. Slack チャンネルを初めて追加する場合は、代わりにを参照してください。Slack 通知を使い始める

   最初のチャンネルを設定したら、この手順に戻って追加のチャンネルを設定してください。
2. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
3. プロジェクトに移動します。
4. ナビゲーションペインで [プロジェクト設定] を選択します。
5. [通知] タブを選択します。
6. [Add channel] (チャンネルの追加) を選択します。
7. 「ワークスペースを選択」を選択し、通知を送信したいチャンネルを含む Slack ワークスペースを選択します。

   Slack ワークスペースがリストにない場合は、の手順に従って追加できます。Slack 通知を使い始める
8. 追加したい Slack チャンネルがプライベートの場合は、チャンネル ID を入力する前に、以下の手順を実行してください。

   a. Slack チャンネルのメッセージボックスに「aws app」**@aws** と入力し、ポップアップから「aws app」を選択します。

b. [Enter] キーを押します。

AWSチャットボットがプライベートチャンネルにいないことを示す Slackbot メッセージが表示されます。

c. 「招待する」を選択して、AWS Chatbot チャンネルに招待します。

9. CodeCatalyst「チャンネル ID」フィールドに Slack チャンネル ID を入力します。ID を見つけるには Slack に移動し、ナビゲーションペインでチャンネルを右クリックして「チャンネルの詳細を開く」を選択します。

チャネル ID はダイアログボックスの下部に表示されます。

10. 「チャンネル名」に名前を入力します。Slack のチャンネル名を使用することをおすすめします。
11. 「通知イベントの選択」で、通知を受け取りたいイベントの種類を選択します。
12. 「追加」 を選択します。

通知チャンネルの通知を編集する

通知の送信先チャンネルを変更したり、特定の通知をすべてオフにしたりできます。

通知を編集するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. プロジェクトに移動します。
3. ナビゲーションペインで [プロジェクト設定] を選択します。
4. [通知] タブを選択します。
5. [通知を編集] を選択します。
6. 次のいずれかを実行します。
   - 特定のチャンネルに通知を送信するには、ドロップダウンリストからチャンネルを選択します。
   - 通知をグローバルにオフにするには、通知の横にあるトグルを選択します。
   - 特定のチャンネルへの通知の送信を停止するには、チャンネルの X を選択します。
7. [保存] を選択します。

チャンネルを削除する。

チャンネルを削除するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. プロジェクトに移動します。ナビゲーションペインで [プロジェクト設定] を選択します。
3. 「プロジェクト設定」 ページで「通知」タブを選択します。
4. 削除するチャンネルの横にあるインジケーターを選択し、「チャンネルを削除」を選択します。メッセージが表示されたら、確認ウィンドウで [OK] を選択します。

ブループリントを使用した CodeCatalyst プロジェクトのセットアップ

ブループリントは、 CodeCatalyst プロジェクトのアーキテクチャコンポーネントを表す任意のコードジェネレーターです。コンポーネントは、1 つのファイル内のワークフローから、サンプルコードで完成したプロジェクト全体まで、あらゆるもので構成されます。ブループリントは任意のオプションセットを取得し、それらを使用して、プロジェクトに転送される任意の出力コードセットを生成します。ブループリントが最新のベストプラクティスまたは新しいオプションで更新されると、そのブループリントを含むプロジェクトでコードベースの関連部分を再生成できます。

Amazon CodeCatalyst ブループリントを使用して、ソースリポジトリ、サンプルソースコード、CI/CD ワークフロー、ビルドレポートとテストレポート、統合された問題追跡ツールを使用して完全なプロジェクトを作成できます。 CodeCatalyst ブループリントは、設定された設定パラメータに基づいてリソースとソースコードを生成します。 CodeCatalyst管理のブループリントを使用する場合、選択したブループリントによってプロジェクトに追加されるリソースと、 が作成または設定する CodeCatalyst ツールが決まり、プロジェクトリソースを追跡して使用できます。ブループリントユーザーは、ブループリントを使用してプロジェクトを作成するか、既存の CodeCatalyst プロジェクトに適用できます。プロジェクトに複数のブループリントを適用でき、それぞれを独立したコンポーネントとして適用できます。例えば、ウェブアプリケーションのブループリントで作成されたプロジェクトを作成し、後でセキュリティブループリントを適用できます。設計図のいずれかが更新されると、ライフサイクル管理を通じてプロジェクトに変更や修正を組み込むことができます。詳細については、「CodeCatalyst ブループリントを使用した包括的なプロジェクトの作成」および「ブループリントユーザーとしてのライフサイクル管理の使用」を参照してください。

設計図の作成者として、 CodeCatalyst スペースメンバーがプロジェクトリソースを使用するためのカスタム設計図を作成して公開することもできます。カスタムブループリントは、スペースのプロジェクトの指定されたニーズを満たすように開発できます。スペースのブループリントカタログにカスタムブループリントを追加した後、ブループリントを管理し、更新を続行して、スペースのプロジェクトを最新のベストプラクティスに更新できます。詳細については、「でのプロジェクトのカスタムブループリントの標準化 CodeCatalyst」を参照してください。ブループリント SDK とサンプルブループリントを表示するには、オープンソース GitHub リポジトリ「」を参照してください。

トピック

- 設計図を使用したプロジェクトの作成
- プロジェクトにブループリントを適用してリソースを追加する
- プロジェクトからブループリントの関連付けを解除して更新を停止する

- プロジェクトのブループリントバージョンの変更
- プロジェクトのブループリントの説明を編集する
- ブループリントユーザーとしてのライフサイクル管理の使用
- CodeCatalyst ブループリントを使用した包括的なプロジェクトの作成
- でのプロジェクトのカスタムブループリントの標準化 CodeCatalyst
- のブループリントのクォータ CodeCatalyst

設計図を使用したプロジェクトの作成

Amazon ブループリントカタログのブループリントまたはカスタム CodeCatalyst ブループリントを含むチームのスペースカタログを使用して、プロジェクトをすばやく作成できます。設計図に応じて、プロジェクトは特定のリソースで作成されます。詳細については、「設計図を使用したプロジェクトの作成」および「 CodeCatalyst ブループリントを使用した包括的なプロジェクトの作成」を参照してください。

プロジェクトを作成したら、カタログまたはスペースの CodeCatalyst カタログから、カスタムブループリントを使用して、追加のブループリントを CodeCatalyst プロジェクトに適用できます。ブループリントはアーキテクチャコンポーネントを表すため、複数のブループリントをプロジェクトで一緒に使用して、チームのベストプラクティスを組み込むことができます。これにより、進化するコンポーネントに対する最新の変更をプロジェクトが最新であることを確認することもできます。プロジェクトでブループリントを使用する方法の詳細については、「」を参照してくださいブループリントユーザーとしてのライフサイクル管理の使用。

プロジェクトにブループリントを適用してリソースを追加する

プロジェクトに複数のブループリントを適用して、機能コンポーネント、リソース、ガバナンスを組み込むことができます。プロジェクトは、個別のブループリントで個別に管理されるさまざまな要素をサポートできます。ブループリントをプロジェクトに適用すると、リソースを手動で作成し、ソフトウェアコンポーネントを機能させる必要がなくなります。また、要件の進化に合わせてプロジェクトを最新の状態に保つこともできます。プロジェクトでのブループリントの適用の詳細については、「」を参照してくださいブループリントユーザーとしてのライフサイクル管理の使用。

ブループリントの詳細を設定するときに、ブループリントのソースコードを優先サードパーティーリポジトリに保存することもできます。このリポジトリでは、ブループリントを管理し、ライフサイクル管理機能を利用してプロジェクトを最新の状態に保つことができます。詳細については、「で拡張

機能を使用してプロジェクトに機能を追加する CodeCatalyst」および「ブループリントユーザーとしてのライフサイクル管理の使用」を参照してください。

ブループリントをプロジェクトに適用するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. CodeCatalyst コンソールでスペースに移動し、設計図を適用するプロジェクトを選択します。
3. ナビゲーションペインで、設計図 を選択し、設計図 を適用 を選択します。
4. 設計図タブからCodeCatalyst 設計図を選択するか、スペース設計図タブからカスタム設計図を選択し、次へ を選択します。
5. ブループリントの詳細 で、ターゲットバージョンドロップダウンメニューからブループリントバージョンを選択します。最新のカタログバージョンが自動的に選択されます。
6. (オプション) デフォルトでは、設計図によって作成されたソースコードはリポジトリに CodeCatalyst 保存されます。または、ブループリントのソースコードをサードパーティーのリポジトリに保存することもできます。詳細については、「で拡張機能を使用してプロジェクトに機能を追加する CodeCatalyst」を参照してください。

   使用するサードパーティーのリポジトリプロバイダーに応じて、次のいずれかを実行します。

   - GitHub リポジトリ： GitHub アカウントを接続します。

     詳細ドロップダウンメニューを選択し、リポジトリプロバイダーとして を選択し GitHub、設計図によって作成されたソースコードを保存する GitHub アカウントを選択します。

     > **Note**
     >
     > GitHub アカウントへの接続を使用している場合は、アイデンティティと CodeCatalyst アイデンティティ間のアイデンティティマッピングを確立するために、個人接続を作成する必要があります GitHub 。詳細については、「個人用接続」および「個人接続による GitHub リソースへのアクセス」を参照してください。

   - Bitbucket : Bitbucket ワークスペースを接続します。

     詳細ドロップダウンメニューを選択し、リポジトリプロバイダーとして Bitbucket を選択し、設計図によって作成されたソースコードを保存する Bitbucket ワークスペースを選択します。
7. 「設計図の設定」で、設計図パラメータを設定します。設計図によっては、ソースリポジトリに名前を付けるオプションがある場合があります。

8. 現在のブループリントバージョンと更新されたバージョンの違いを確認します。プルリクエストに表示される違いは、現在のバージョンと、プルリクエストの作成時に必要なバージョンである最新バージョンの間の変更を示しています。変更が表示されない場合、バージョンは同じであるか、現在のバージョンと目的のバージョンの両方に同じバージョンを選択している可能性があります。
9. プルリクエストにレビューするコードと変更が含まれていることを確認したら、ブループリントを適用 を選択します。プルリクエストを作成したら、コメントを追加できます。コメントは、プルリクエスト、またはファイル内の個々の行、およびプルリクエスト全体に追加できます。ファイルなどのリソースへのリンクは、 @ 署名の後にファイルの名前を付けることで追加できます。

> **Note**
>
> ブループリントは、プルリクエストが承認されてマージされるまで適用されません。詳細については、「プルリクエストの確認」および「プルリクエストのマージ」を参照してください。

ブループリントの作成者は、新しいプロジェクトを作成したり、既存のプロジェクトに適用したりするために、ブループリントが利用できない指定されたスペースのプロジェクトにカスタムブループリントを適用することもできます。詳細については、「指定されたスペースとプロジェクトでのカスタムブループリントの公開と適用」を参照してください。

ブループリントの更新を受信しなくなった場合は、プロジェクトからブループリントの関連付けを解除できます。詳細については、「プロジェクトからブループリントの関連付けを解除して更新を停止する」を参照してください。

プロジェクトからブループリントの関連付けを解除して更新を停止する

ブループリントから新しい更新を行いたくない場合は、ブループリントとプロジェクトの関連付けを解除できます。設計図からプロジェクトに追加されたリソースと機能ソフトウェアコンポーネントは、プロジェクトに残ります。

プロジェクトからブループリントの関連付けを解除するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。

2. CodeCatalyst コンソールでスペースに移動し、設計図の関連付けを解除するプロジェクトを選択します。
3. ナビゲーションペインで [Blueprints] (ブループリント) を選択します。
4. 関連付けを解除するリソースを含むブループリントを選択し、アクションドロップダウンメニューを選択し、ブループリントの関連付けを解除を選択します。
5. `confirm` を入力して、関連付けが解除されたことを確認します。
6. [確認] を選択します。

プロジェクトのブループリントバージョンの変更

設計図を使用してプロジェクトを作成した場合、または既存のプロジェクトに設計図を適用した場合、設計図の新しいバージョンが通知されます。承認されたプルリクエストによってブループリントバージョンが更新される前に、コードの変更と影響を受ける環境を表示できます。ライフサイクル管理では、プロジェクトに適用された 1 つ以上のブループリントのバージョンを変更できるため、各ブループリントバージョンはプロジェクトの他の領域に影響を与えることなく変更できます。設計図の更新を上書きすることもできます。詳細については、「ブループリントユーザーとしてのライフサイクル管理の使用」を参照してください。

ブループリントを最新バージョンに更新するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. CodeCatalyst コンソールで、ブループリントのバージョンを更新するスペースに移動します。
3. スペースダッシュボードで、更新するブループリントを含むプロジェクトを選択します。
4. ナビゲーションペインで [Blueprints] (ブループリント) を選択します。
5. ステータス 列で、カタログバージョンを変更するリンクを選択します (カタログバージョン 0.3.109 の変更など)。
6. Actions ドロップダウンメニューを選択し、バージョンの更新 を選択します。最新バージョンが自動的に選択されます。
7. (オプション) 設計図 の設定 で、設計図パラメータを設定します。
8. (オプション)「コード変更」タブで、現在のブループリントバージョンと更新されたバージョンの違いを確認します。プルリクエストに表示される違いは、現在のバージョンと最新バージョンの間の変更です。これは、プルリクエストの作成時に必要なバージョンです。変更が表示されない場合、バージョンは同じであるか、現在のバージョンと目的のバージョンの両方に同じバージョンを選択している可能性があります。

9. プルリクエストにレビューするコードと変更が含まれていることを確認したら、更新を適用 を選択します。プルリクエストを作成したら、コメントを追加できます。コメントは、プルリクエスト、またはファイル内の個々の行、およびプルリクエスト全体に追加できます。ファイルなどのリソースへのリンクは、 @ 署名の後にファイルの名前を付けることで追加できます。

> **Note**
>
> プルリクエストが承認されてマージされるまで、ブループリントは更新されません。詳細については、「プルリクエストの確認」および「プルリクエストのマージ」を参照してください。

> **Note**
>
> ブループリントバージョンの更新のために既存のプルリクエストが開いている場合は、新しいプルリクエストを作成する前に以前のプルリクエストを閉じます。バージョンの更新を選択すると、ブループリントの保留中のプルリクエストのリストが表示されます。プロジェクト設定およびプロジェクト概要ページのブループリントタブから保留中のプルリクエストを表示することもできます。詳細については、「プルリクエストの表示」を参照してください。

ブループリントのバージョンを変更するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. CodeCatalyst コンソールで、ブループリントのバージョンを更新するスペースに移動します。
3. スペースダッシュボードで、更新するブループリントを含むプロジェクトを選択します。
4. ナビゲーションペインで、ブループリント を選択し、更新するブループリントのラジオボタンを選択します。
5. Actions ドロップダウンメニューを選択し、設計図 の設定 を選択します。
6. ターゲットバージョンドロップダウンメニューから、使用するバージョンを選択します。最新バージョンが自動的に選択されます。
7. (オプション) 設計図 の設定 で、設計図パラメータを設定します。
8. (オプション)「コード変更」タブで、現在のブループリントバージョンと更新されたバージョンの違いを確認します。プルリクエストに表示される違いは、現在のバージョンと最新バージョンの間の変更です。これは、プルリクエストの作成時に必要なバージョンです。変更が表示され

ない場合、バージョンは同じであるか、現在のバージョンと目的のバージョンの両方に同じバージョンを選択している可能性があります。

9. プルリクエストにレビューするコードと変更が含まれていることを確認したら、更新を適用 を選択します。プルリクエストを作成したら、コメントを追加できます。コメントは、プルリクエスト、またはファイル内の個々の行、およびプルリクエスト全体に追加できます。ファイルなどのリソースへのリンクは、 @ 署名の後にファイルの名前を付けることで追加できます。

> **Note**
>
> プルリクエストが承認されてマージされるまで、ブループリントは更新されません。詳細については、「プルリクエストの確認」および「プルリクエストのマージ」を参照してください。

> **Note**
>
> ブループリントバージョンの更新のために既存のプルリクエストが開いている場合は、新しいプルリクエストを作成する前に以前のプルリクエストを閉じます。バージョンの更新を選択すると、ブループリントの保留中のプルリクエストのリストが表示されます。プロジェクト設定およびプロジェクト概要ページのブループリントタブから保留中のプルリクエストを表示することもできます。詳細については、「プルリクエストの表示」を参照してください。

プロジェクトのブループリントの説明を編集する

プロジェクトの作成に使用した、またはプロジェクトの作成後に適用した設計図の説明を編集できます。設計図は、プロジェクトで複数回使用できます。プロジェクト内のブループリントの目的を区別するために、それらのブループリントの説明を使用できます。説明は、特定のブループリントから適用するコンポーネントを識別するためにも使用できます。

プロジェクトでブループリントの説明を編集するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. CodeCatalyst コンソールでスペースに移動し、更新するブループリント設定でプロジェクトを選択します。
3. ナビゲーションペインで [Blueprints] (ブループリント) を選択します。

4. 更新する説明を含むブループリントを選択し、アクションドロップダウンメニューを選択し、説明の編集を選択します。
5. 設計図の説明テキスト入力フィールドに、プロジェクト内の設計図を識別するための説明を入力します。
6. [保存] を選択します。

ブループリントユーザーとしてのライフサイクル管理の使用

ライフサイクル管理は、更新されたオプションまたは設計図のバージョンからコードベースを再生成する機能です。これにより、設計図の作成者は、特定の設計図を含むすべてのプロジェクトのソフトウェア開発ライフサイクルを一元管理できます。例えば、ウェブアプリケーションブループリントにセキュリティ修正をプッシュすると、ウェブアプリケーションブループリントを含む、またはウェブアプリケーションブループリントから作成されたすべてのプロジェクトで、その修正を自動的に取得できるようになります。この同じ管理フレームワークにより、設計図ユーザーは、設計図オプションを選択した後に変更することもできます。

トピック

既存のプロジェクトでのライフサイクル管理の使用

ライフサイクル管理は、ブループリントから作成されたプロジェクト、またはブループリントに関連付けられていない既存のプロジェクトに使用できます。例えば、設計図から一度も作成されなかった five-year-old Java アプリケーションに、標準のセキュリティプラクティスの設計図を追加できます。設計図は、セキュリティスキャンワークフローおよびその他の関連コードを生成します。これで、Java アプリケーションのコードベースのその部分は、設計図に変更が加えられるたびに、チームのベストプラクティスに合わせて自動的に最新の状態になります。

プロジェクト内の複数のブループリントでのライフサイクル管理の使用

設計図はアーキテクチャコンポーネントを表すため、複数の設計図を同じプロジェクトで一緒に使用することがよくあります。例えば、プロジェクトは、企業のプラットフォームエンジニアによって構築された中央ウェブ API ブループリントと、アプリケーションセキュリティチームが構築したリリースチェックブループリントで構成できます。これらの設計図はそれぞれ個別に更新でき、過去に適用されたマージ解決が記憶されます。

> **Note**
>
> 任意のアーキテクチャコンポーネントとして、すべてのブループリントが互いに結合しようとしても、一緒に意味をなしたり、論理的に一緒に動作したりするわけではありません。

ライフサイクルプルリクエストでの競合の操作

場合によっては、ライフサイクルプルリクエストによってマージの競合が発生することがあります。これらは手動で解決できます。解決策は、その後の設計図の更新時に記憶されます。

ライフサイクル管理の変更のオプトアウト

ユーザーはプロジェクトからブループリントを削除して、ブループリントへのすべての参照の関連付けを解除し、ライフサイクル更新をオプトアウトできます。安全上の理由から、設計図から追加されたものを含め、プロジェクトのコードやリソースを削除したり、影響を与えたりすることはありません。詳細については、「プロジェクトからブループリントの関連付けを解除して更新を停止する」を参照してください。

プロジェクトでのブループリントのライフサイクル管理の上書き

プロジェクト内の特定のファイルに対する設計図の更新を上書きする場合は、リポジトリに所有権ファイルを含めることができます。GitLabのコード所有者仕様が推奨ガイドラインです。設計図では、常に他のものよりもコード所有者ファイルが優先され、次のようなサンプルを生成できます。

```
new BlueprintOwnershipFile(sourceRepo, {
      resynthesis: {
        strategies: [
          {
            identifier: 'dont-override-sample-code',
```

```
            description: 'This strategy is applied accross all sample code. The
 blueprint will create sample code, but skip attempting to update it.',
            strategy: MergeStrategies.neverUpdate,
            globs: [
              '**/src/**',
              '**/css/**',
            ],
          },
        ],
      },
    });
```

これにより、次の内容.ownership-fileの が生成されます。

```
[dont-override-sample-code] @amazon-codecatalyst/blueprints.import-from-git
# This strategy is applied accross all sample code. The blueprint will create sample
 code, but skip attempting to update it.
# Internal merge strategy: neverUpdate
**/src/**
**/css/**
```

CodeCatalyst ブループリントを使用した包括的なプロジェクトの作成

ブループリントを使用してプロジェクトを作成すると、 はソースリポジトリ、サンプルソースコード、CI/CD ワークフロー、ビルドレポートとテストレポート、統合された問題追跡ツールを使用して完全なプロジェクト CodeCatalyst を作成します。プロジェクトのブループリントでは、コードを使用して、さまざまなタイプのアプリケーションやフレームワークのクラウドインフラストラクチャ、リソース、サンプルソースアーティファクトをプロビジョニングします。

詳細については、「「プロジェクトの作成」」を参照してください。プロジェクトを作成するには、スペース管理者である必要があります。

トピック

- 使用可能なブループリント
- プロジェクトのブループリント情報の検索

使用可能なブループリント

設計図名	設計図の説明
ASP.NET Core ウェブ API	このブループリントは、.NET 6 ASP.NET Core ウェブ API アプリケーションを作成します。設計図では、.NET 用の AWS デプロイツールを使用し、Amazon Elastic Container Service を設定するオプション AWS App Runner、またはデプロイターゲット AWS Elastic Beanstalk として オプションを提供します。
AWS Glue ETL	この設計図では、 AWS CDK、 AWS Glue、AWS Lambda、および Amazon Athena を使用してサンプル抽出変換負荷 (ETL) リファレンス実装を作成し、カンマ区切り値 (CSVsを Apache Parquet に変換します。
DevOps デプロイパイプライン	この設計図では、複数のステージに AWS またがってリファレンスアプリケーションを にデプロイする AWS Deployment Pipeline リファレンスアーキテクチャを使用してデプロイパイプラインを作成します。
を使用した Java API AWS Fargate	この設計図は、コンテナ化されたウェブサービスプロジェクトを作成します。このプロジェクトでは、AWS Copilot CLI を使用して、Amazon ECS 上の Amazon DynamoDB にバックアップされたコンテナ化された Spring Boot Java ウェブサービスを構築およびデプロイします。プロジェクトは、 AWS Fargate サーバーレスコンピューティング上の Amazon ECS クラスターにコンテナ化されたアプリケーションをデプロイします。アプリケーションは DynamoDB テーブルにデータを保存します。ワークフローが正常に実行されると、

設計図名	設計図の説明
	サンプルウェブサービスは Application Load Balancer を通じて公開されます。
最新の 3 層ウェブアプリケーション	このブループリントは、Python でアプリケーションレイヤーと Vue フロントエンドフレームワークのコードを生成し、適切に設計された 3 層の最新ウェブアプリケーションを構築してデプロイします。
.NET サーバーレスアプリケーション	この設計図では、.NET CLI Lambda ツールを使用して AWS Lambda 関数を作成します。設計図には、C# または F# の選択など、 AWS Lambda 関数のオプションが用意されています。
を使用した Node.js API AWS Fargate	この設計図は、コンテナ化されたウェブサービスプロジェクトを作成します。このプロジェクトでは、AWS Copilot CLI を使用して、コンテナ化された Express/Node.js ウェブサービスを Amazon Elastic Container Service で構築およびデプロイします。プロジェクトは、 AWS Fargate サーバーレスコンピューティング上の Amazon ECS クラスターにコンテナ化されたアプリケーションをデプロイします。ワークフローが正常に実行されると、サンプルウェブサービスは Application Load Balancer を通じて公開されます。
サーバーレスアプリケーションモデル (SAM)	このブループリントは、サーバーレスアプリケーションモデル (SAM) を使用して API を作成およびデプロイするプロジェクトを作成します。SDK for Java TypeScript、または SDK for Python をプログラミング言語として選択できます。

設計図名	設計図の説明
サーバーレスイメージハンドラー	このブループリントは、画質を低下させることなく、高速画像処理用のアプリケーションを作成します。
サーバーレス RESTful マイクロサービス	このブループリントは、To Do サービスリファレンス Amazon API Gateway で AWS Lambda と を使用する REST API を作成します。SDK for Java、 TypeScript、または SDK for Python をプログラミング言語として選択できます。
単一ページアプリケーション	このブループリントは、React、Vue、および Angular フレームワークを使用する単一ページアプリケーション (SPA) を作成します。ホスティングの場合は、 AWS Amplify ホスティングまたは Amazon CloudFront と Amazon S3 から選択します。
静的ウェブサイト	このブループリントは、Hugo または Jekyll 静的サイトジェネレーターを使用して静的ウェブサイトを作成します。静的サイトジェネレーターは、テキスト入力ファイル (Markdown など) を使用して静的ウェブページを生成します。これらは、製品ページ、ドキュメント、ブログなど、ほとんど変更されない情報コンテンツに最適です。設計図では、 を使用して AWS CDK 静的ウェブページを AWS Amplify または Amazon S3 + のいずれかにデプロイします CloudFront。
ウェブアプリケーションを実行するには	この設計図では、フロントエンドコンポーネントとバックエンドコンポーネントを使用して To Do サーバーレスウェブアプリケーションを作成します。SDK for Java TypeScript、または SDK for Python をプログラミング言語として選択できます。

設計図名	設計図の説明
V ideo-on-demand ウェブサービス	このブループリントは、コンテンツを取得、トランスコード、配信する機能を提供する video-on-demand サービスを作成します。設計図では、 AWS Lambda、Amazon S3 Amazon CloudWatch、および を使用します AWS Elemental MediaConvert。
外部ブループリントをサブスクライブする	このブループリントは、インポートされたパッケージごとにワークフローを作成します。これらのワークフローは 1 日に 1 回実行され、NPM でパッケージの新しいバージョンを確認します。新しいバージョンが存在する場合、ワークフローはそれをカスタムブループリントとして CodeCatalyst スペースに追加しようとします。パッケージが見つからないか、設計図でない場合、アクションは失敗します。ターゲットパッケージは NPM 上にあり、パッケージは設計図である必要があります。スペースは、カスタムブループリントをサポートする階層でサブスクライブする必要があります。
Bedrock GenAI チャットボット	このブループリントは、Amazon Bedrock と Anthropic の Claude を使用して生成 AI チャットボットを作成します。このブループリントを使用すると、データに合わせてカスタマイズできる、ログインで保護された独自の LLM プレイグラウンドを構築してデプロイできます。詳細については、「Bedrock GenAI Chatbot ドキュメント」を参照してください。

設計図名	設計図の説明
AWS プロジェクト開発キット (AWS PDK) の設計図	これらの PDK ブループリントは、React ウェブサイト、Smithy API、およびそれを AWS にデプロイするためのサポート CDK インフラストラクチャで構成されるアプリケーションを作成するために一緒に構成できます。AWS PDK は、一般的なパターンの構成要素と、プロジェクトを管理および構築するための開発ツールを提供します。詳細については、「AWS PDK GitHub ソースリポジトリ」および「」を参照してくださいチュートリアル: 構成可能な PDK ブループリントを使用したフルスタックアプリケーションの作成。

プロジェクトのブループリント情報の検索

いくつかのプロジェクトブループリントが で利用できます CodeCatalyst。設計図ごとに、概要と README ファイルが添付されています。概要では、ブループリントによってインストールされるリソースについて説明し、README ファイルではブループリントの詳細と使用方法を説明します。

でのプロジェクトのカスタムブループリントの標準化 CodeCatalyst

カスタムブループリントを使用して、 CodeCatalyst スペースのプロジェクトの開発とベストプラクティスを標準化できます。カスタムブループリントを使用して、ワークフロー定義やアプリケーションコードなど、 CodeCatalyst プロジェクトのさまざまな側面を定義できます。カスタムブループリントを使用して新しいプロジェクトを作成するか、既存のプロジェクトに適用した後、ブループリントへの変更はプルリクエストの更新としてそれらのプロジェクトで使用できます。ブループリントの作成者は、スペース全体でブループリントを使用しているプロジェクトの詳細を表示できるため、プロジェクト間で標準がどのように適用されているかを確認できます。ブループリントのライフサイクル管理により、すべてのプロジェクトのソフトウェア開発ライフサイクルを一元管理できるため、スペース内のプロジェクトが最新の変更または修正でベストプラクティスに従っていることを確認できます。詳細については、「ブループリントの作成者としてライフサイクル管理を使用する」を参照してください。

カスタムブループリントは、再合成を通じて以前のプロジェクトに対してブループリントバージョンを更新する機能を提供します。再合成とは、更新されたバージョンでブループリント合成を再実行するプロセス、または修正や変更を既存のプロジェクトに組み込む機能です。詳細については、「カスタムブループリントの概念」を参照してください。

ブループリント SDK とサンプルブループリントを表示するには、オープンソース GitHub リポジトリ「」を参照してください。

トピック

カスタムブループリントの概念

ここでは、 でカスタムブループリントを使用する際に知っておくべき概念と用語をいくつか紹介します CodeCatalyst。

トピック

- 再合成
- 部分的なオプション
- Projen

ブループリントプロジェクト

ブループリントプロジェクトを使用すると、ブループリントを開発してスペースに公開できます。ソースリポジトリはプロジェクト作成プロセス中に作成され、リポジトリの名前はプロジェクトリソースの詳細を入力するときに選択したものです。ブループリント作成プロセス中にワークフローリリースを生成することを選択した場合、ブループリントビルダーブループリントを使用して公開ワークフローがブループリントに作成されます。ワークフローは最新バージョンを自動的に公開します。

スペースの設計図

スペースのブループリントセクションに移動すると、スペースブループリントテーブルからすべてのブループリントを表示および管理できます。ブループリントがスペースに公開されると、スペースブループリントとして利用可能になり、スペースのブループリントカタログに追加および削除されます。スペースの「ブループリント」セクションで、公開アクセス許可を管理したり、 からブループリントを削除したりすることもできます。詳細については、「カスタムブループリントの詳細、バージョン、プロジェクトの表示」を参照してください。

スペースブループリントカタログ

スペースのブループリントカタログから、追加されたすべてのカスタムブループリントを表示できます。ここでは、スペースメンバーがカスタムブループリントを選択して新しいプロジェクトを作成できます。このカタログは、すべてのスペースメンバーに利用可能なブループリントがすでに存在する CodeCatalyst カタログとは異なります。詳細については、「 CodeCatalyst ブループリントを使用した包括的なプロジェクトの作成」を参照してください。

合成

合成は、 CodeCatalyst プロジェクト内のソースコード、設定、およびリソースを表すプロジェクトバンドルを生成するプロセスです。その後、バンドルは CodeCatalyst デプロイ API オペレーションによってプロジェクトにデプロイされるために使用されます。このプロセスは、カスタムブループリントの開発中にローカルで実行できるため、 でプロジェクトを作成しなくてもプロジェクトの作成をエミュレートできます CodeCatalyst。合成を実行するには、次のコマンドを使用できます。

```
yarn blueprint:synth            # fast mode
```

```
yarn blueprint:synth --cache     # wizard emulation mode
```

ブループリントは、 にマージされたオプションを使用してメインblueprint.tsクラスを呼び出すことによって開始されますdefaults.json。新しいプロジェクトバンドルが synth/synth.*[options-name]*/proposed-bundle/フォルダの下に生成されます。出力には、設定したオプションを含む、カスタムブループリントが生成???するプロジェクトバンドルが含まれます。

再合成

再合成は、さまざまな設計図オプションまたは既存のプロジェクトの設計図バージョンを使用して設計図を再生成するプロセスです。設計図の作成者は、カスタム設計図コードでカスタムマージ戦略を定義できます。で所有権の境界を定義.ownership-fileして、設計図を更新できるコードベースの部分を指定することもできます。カスタムブループリントは の更新を提案できますが.ownership-file、カスタムブループリントを使用するプロジェクトデベロッパーは、プロジェクトの所有権の境界を決定できます。再合成をローカルで実行し、カスタムブループリントを公開する前にテストして更新できます。再合成を実行するには、次のコマンドを使用します。

```
yarn blueprint:resynth             # fast mode
yarn blueprint:resynth --cache     # wizard emulation mode
```

ブループリントは、 にマージされたオプションを使用してメインblueprint.tsクラスを呼び出すことによって開始されますdefaults.json。新しいプロジェクトバンドルが synth/resynth.*[options-name]*/フォルダの下に生成されます。出力には、設定したオプションと、設定した可能性のある部分オプションを考慮して、カスタムブループリントが生成するプロジェクトバンドルが含まれます。

合成および再合成プロセスの後に、次のコンテンツが作成されます。

- proposed-bundle - ターゲットブループリントバージョンの新しいオプションを使用して実行したときの合成の出力。
- existing-bundle - 既存のプロジェクトのモック。このフォルダに何もない場合は、 と同じ出力で生成されますproposed-bundle。
- 祖先バンドル - 以前のバージョン、以前のオプション、または組み合わせで実行した場合にブループリントが生成する内容のモック。このフォルダに何もない場合は、 と同じ出力で生成されますproposed-bundle。
- resolved-bundle - バンドルは常に再生成され、デフォルトで proposed-bundle、existing-bundle、および 間の 3 方向マージになりますancestor-bundle。このバンドルは、再合成がローカルで出力される内容をエミュレーションします。

ブループリント出力バンドルの詳細については、「」を参照してください再合成によるファイルの生成。

部分的なオプション

Options インターフェイス全体を列挙する必要src/wizard-configuration/のない にオプションバリエーションを追加でき、オプションは defaults.json ファイルの上にマージされます。これにより、特定のオプション間でテストケースを調整できます。

例:

Options インターフェイス :

```
{
  language: "Python" | "Java" | "Typescript",
  repositoryName: string
  ...
}
```

defaults.json ファイル:

```
{
  language: "Python",
  repositoryName: "Myrepo"
  ...
}
```

追加の設定テスト :

- ```
  #wizard-config-typescript-test.json
  {
    language: "Typescript",
  }
  ```
- ```
  #wizard-config-java-test.json
  {
    language: "Java",
  }
  ```

Projen

Projen は、カスタムブループリントがそれ自体を最新で一貫性のある状態に維持するために使用するオープンソースツールです。ブループリントは Projen パッケージとして提供されます。このフレームワークでは、プロジェクトを構築、バンドル、公開でき、 インターフェイスを使用してプロジェクトの設定と設定を管理できるためです。

Projen を使用すると、作成後でもブループリントを大規模に更新できます。Projen ツールは、プロジェクトバンドルを生成するブループリント合成の基盤となるテクノロジーです。Projen はプロジェクトの設定を所有しており、設計図の作成者としてユーザーに影響を与えることはできません。`yarn projen` を実行して、依存関係を追加した後にプロジェクトの設定を再生成するか、`projenrc.ts` ファイル内のオプションを変更できます。Projen は、プロジェクトを合成するためのカスタムブループリントの基盤となる生成ツールでもあります。詳細については、「projen GitHub page」を参照してください。Projen の操作の詳細については、Projen のドキュメントおよび Projen を使用してプロジェクト設定を簡素化する方法を参照してください。

カスタムブループリントの開始方法

ブループリントを作成するプロセス中に、ブループリントを設定し、プロジェクトリソースのプレビューを生成できます。各カスタムブループリントはプロジェクトによって管理されます。CodeCatalyst プロジェクトには、スペースのブループリントカタログに公開するためのワークフローがデフォルトで含まれています。

カスタムブループリントの詳細を設定するときに、ブループリントのソースコードをサードパーティーリポジトリに保存することもできます。このリポジトリでは、カスタムブループリントを管理し、ライフサイクル管理機能を利用して、カスタムブループリントが変更されてもスペースのプロジェクトの同期を維持できます。詳細については、「で拡張機能を使用してプロジェクトに機能を追加する CodeCatalyst」および「ブループリントの作成者としてライフサイクル管理を使用する」を参照してください。

トピック

- 前提条件
- ステップ 1: でカスタムブループリントを作成する CodeCatalyst
- ステップ 2: コンポーネントを使用してカスタムブループリントを作成する
- ステップ 3: カスタムブループリントをプレビューする
- (オプション) ステップ 4: カスタムブループリントプレビューバージョンを公開する

前提条件

カスタムブループリントを作成する前に、次の要件を考慮してください。

- CodeCatalyst スペースはエンタープライズ階層である必要があります。詳細については、「Amazon CodeCatalyst 管理者ガイド」の「請求の管理」を参照してください。
- カスタムブループリントを作成するには、スペース管理者またはパワーユーザーロールが必要です。詳細については、「ユーザーロールによるアクセス許可の付与」を参照してください。

ステップ 1: でカスタムブループリントを作成する CodeCatalyst

スペースの設定からカスタムブループリントを作成すると、リポジトリが自動的に作成されます。リポジトリには、スペースのブループリントカタログに公開する前にブループリントを開発するために必要なすべてのリソースが含まれています。

カスタムブループリントを作成するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. CodeCatalyst コンソールで、カスタムブループリントを作成するスペースに移動します。
3. スペースダッシュボードで、設定タブを選択し、ブループリント を選択します。
4. ブループリントの作成 を選択します。
5. 「設計図の名前」に、プロジェクトに割り当てる名前と、それに関連するリソース名を入力します。名前はスペース内で一意でなければなりません。
6. (オプション) デフォルトでは、ブループリントによって作成されたソースコードはリポジトリに CodeCatalyst 保存されます。または、ブループリントのソースコードをサードパーティーのリポジトリに保存することもできます。詳細については、「で拡張機能を使用してプロジェクトに機能を追加する CodeCatalyst」を参照してください。

   使用するサードパーティーのリポジトリプロバイダーに応じて、次のいずれかを実行します。

   - GitHub リポジトリ : GitHub アカウントを接続します。

     詳細ドロップダウンメニューを選択し、リポジトリプロバイダーとして を選択し GitHub、設計図によって作成されたソースコードを保存する GitHub アカウントを選択します。

> **Note**
>
> GitHub アカウントへの接続を使用している場合は、アイデンティティと CodeCatalyst アイデンティティ間のアイデンティティマッピングを確立するために、個人接続を作成する必要があります GitHub 。詳細については、「個人用接続」および「個人接続による GitHub リソースへのアクセス」を参照してください。

- Bitbucket : Bitbucket ワークスペースを接続します。

  詳細ドロップダウンメニューを選択し、リポジトリプロバイダーとして Bitbucket を選択し、設計図によって作成されたソースコードを保存する Bitbucket ワークスペースを選択します。

7. 設計図の詳細 で、次の操作を行います。

   a. 設計図の表示名のテキスト入力フィールドに、スペースの設計図カタログに表示される名前を入力します。

   b. 説明テキスト入力フィールドに、カスタムブループリントの説明を入力します。

   c. 作成者名のテキスト入力フィールドに、カスタム設計図の作成者名を入力します。

   d. （オプション) 詳細設定 を選択します。

      i. + 追加 を選択して、`package.json`ファイルに追加されるタグを追加します。

      ii. ライセンスドロップダウンメニューを選択し、カスタムブループリントのライセンスを選択します。

      iii. 設計図パッケージ名のテキスト入力フィールドに、設計図パッケージを識別する名前を入力します。

      iv. デフォルトでは、リリースワークフローは、Blueprint Builder と呼ばれるプロジェクト内の公開ブループリントを使用して生成されます。リリースワークフローで公開許可が有効になっているため、ワークフローは変更をプッシュすると、最新のブループリントバージョンをスペースに公開します。ワークフロー生成をオフにするには、「ワークフローのリリース」チェックボックスをオフにします。

8. （オプション) ブループリントプロジェクトには、スペースカタログへのブループリントの公開をサポートする定義済みコードが付属しています。行ったプロジェクトパラメータの選択に基づいて更新を含む定義ファイルを表示するには、「設計図プレビューの生成」から「コードを表示」または「ワークフローを表示」を選択します。

9. ブループリントの作成 を選択します。

カスタムブループリントのワークフロー生成をオフにしなかった場合、ブループリントの作成時にワークフローが自動的に実行され始めます。ワークフローの実行が完了すると、デフォルトでカスタムブループリントをスペースのブループリントカタログに追加できます。最新のブループリントバージョンをスペースに自動的に公開したくない場合は、公開アクセス許可をオフにできます。詳細については、「カスタムブループリントの発行許可の設定」および「ワークフローの実行」を参照してください。

という発行ワークフローblueprint-releaseは設計図を使用して作成されるため、設計図はプロジェクトに適用された設計図として確認できます。詳細については、「プロジェクトにブループリントを適用してリソースを追加する」および「プロジェクトからブループリントの関連付けを解除して更新を停止する」を参照してください。

ステップ 2: コンポーネントを使用してカスタムブループリントを作成する

ブループリントウィザードは、カスタムブループリントの作成時に生成され、カスタムブループリントの開発時にコンポーネントで変更できます。`src/blueprints.js` および `src/defaults.json` ファイルを更新してウィザードを変更できます。

> **Important**
>
> 外部ソースからのブループリントパッケージを使用する場合は、それらのパッケージに伴うリスクを考慮してください。スペースに追加するカスタムブループリントとそれらが生成するコードは、お客様の責任となります。

設計図コードを設定する前に、サポートされている統合開発環境 (IDE) を使用して CodeCatalyst プロジェクトに開発環境を作成します。開発環境は、必要なツールとパッケージを操作するために必要です。

開発環境を作成するには

1. ナビゲーションペインで、次のいずれかを実行します。
   a. 概要 を選択し、「開発環境」セクションに移動します。
   b. コード を選択し、開発環境 を選択します。
   c. コード を選択し、ソースリポジトリ を選択し、ブループリントの作成時に作成したリポジトリを選択します。
2. [開発環境を作成] を選択します。

3. ドロップダウンメニューからサポートされている IDE を選択します。詳細については、「開発環境でサポートされている統合開発環境」を参照してください。
4. 「既存のブランチで作業」を選択し、「既存のブランチ」ドロップダウンメニューから、作成した機能ブランチを選択します。
5. (オプション) エイリアス - オプションのテキスト入力フィールドに、エイリアスを入力して開発環境を識別します。
6. [作成] を選択します。開発環境の作成中に、開発環境のステータス列に開始 と表示され、開発環境の作成時にステータス列に実行中と表示されます。

詳細については、「で開発環境を使用してコードを記述および変更する CodeCatalyst」を参照してください。

カスタムブループリントを開発するには

1. 作業ターミナルで、次のyarnコマンドを使用して依存関係をインストールします。

   ```
   yarn
   ```

   必要なツールとパッケージは、Yarn を含む CodeCatalyst 開発環境を通じて利用できます。開発環境なしでカスタムブループリントを使用する場合は、まずシステムに Yarn をインストールします。詳細については、「Yarn のインストールドキュメント」を参照してください。
2. カスタムブループリントを開発して、設定に合わせて設定します。コンポーネントを追加することで、ブループリントのウィザードを変更できます。詳細については、プロジェクト要件を満たすためのカスタムブループリントの開発、フロントエンドウィザードを使用した設計図機能の変更、およびスペースへのカスタムブループリントの発行を参照してください。

ステップ 3: カスタムブループリントをプレビューする

カスタムブループリントを設定して開発したら、ブループリントのプレビューバージョンをプレビューしてスペースに公開できます。プレビューバージョンを使用すると、ブループリントが新しいプロジェクトの作成に使用されたり、既存のプロジェクトに適用されたりする前に、ブループリントが目的のものであることを確認することができます。

カスタムブループリントをプレビューするには

1. 作業ターミナルで、次のyarnコマンドを使用します。

```
yarn blueprint:preview
```

2. 提供されているSee this blueprint at:リンクに移動して、カスタムブループリントをプレビューします。
3. 設定に基づいて、テキストを含む UI が想定どおりに表示されることを確認します。カスタムブループリントを変更する場合は、blueprint.tsファイルを編集し、ブループリントを再合成してから、プレビューバージョンを再度公開できます。詳細については、「再合成」を参照してください。

(オプション) ステップ 4: カスタムブループリントプレビューバージョンを公開する

カスタムブループリントのプレビューバージョンをスペースのブループリントカタログに追加する場合は、スペースに公開できます。これにより、非プレビューバージョンをカタログに追加する前に、ブループリントをユーザーとして表示できます。プレビューバージョンでは、実際のバージョンを使用せずに公開できます。例えば、ある0.0.1バージョンで作業する場合、プレビューバージョンを公開して追加できるため、2 番目のバージョンに対する新しい更新を公開して として追加できます0.0.2。

カスタムブループリントのプレビューバージョンを公開するには

提供されているEnable version *[version number]* at:リンクに移動して、カスタムブループリントを有効にします。このリンクは、 で yarn コマンドを実行するときに提供されますステップ 3: カスタムブループリントをプレビューする。

カスタムブループリントを作成、開発、プレビュー、公開した後、スペースのブループリントカタログに最終的なブループリントバージョンを公開して追加できます。詳細については、「スペースへのカスタムブループリントの発行」および「スペースカタログへのカスタムブループリントの追加」を参照してください。

チュートリアル:React アプリケーションの作成と更新

ブループリント作成者は、カスタムブループリントを作成してスペースのブループリントカタログに追加できます。これらのブループリントは、スペースメンバーが新しいプロジェクトを作成したり、既存のプロジェクトに適用したりするために使用できます。ブループリントには引き続き変更を加え、プルリクエストを通じて更新することができます。

このチュートリアルでは、ブループリント作成者の視点とブループリントユーザーの視点からウォークスルーを説明します。このチュートリアルでは、React の単一ページのウェブアプリケーションブ

ループリントを作成する方法を説明します。その後、そのブループリントを使って新しいプロジェクトを作成します。ブループリントが変更で更新されると、ブループリントから作成されたプロジェクトはプルリクエストを通じてそれらの変更を組み込みます。

トピック

前提条件

カスタムブループリントを作成および更新するには、以下のタスクを完了している必要があります。のセットアップと へのサインイン CodeCatalyst

- AWS CodeCatalystサインインするためのビルダー ID を持っている。
- スペースに所属していて、そのスペースのスペース管理者またはパワーユーザーロールが割り当てられていること。詳細については、スペースの作成、ユーザーにスペース許可を付与する、およびスペース管理者ロールを参照してください。

ステップ 1: カスタムブループリントを作成する

カスタムブループリントを作成すると、 CodeCatalyst ブループリントのソースコードと開発ツールとリソースを含むプロジェクトが作成されます。プロジェクトは、ブループリントを開発、テスト、公開する場所です。

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. CodeCatalyst コンソールで、ブループリントを作成したいスペースに移動します。
3. [設定] を選択してスペース設定に移動します。

4. 「スペース設定」タブで「ブループリント」を選択し、「ブループリントを作成」を選択します。
5. ブループリント作成ウィザードのフィールドを以下の値で更新します。

   - [ブループリント名] に、と入力します。`react-app-blueprint`
   - [ブループリント表示名] に、と入力します。`react-app-blueprint`
6. オプションで [コードを表示] を選択し、ブループリントのブループリントソースコードをプレビューします。同様に、[ワークフローを表示] を選択すると、ブループリントをビルドして公開するプロジェクトで作成されるワークフローをプレビューできます。
7. [ブループリントを作成] を選択します。
8. ブループリントが作成されると、ブループリントのプロジェクトに移動します。このプロジェクトには、ブループリントのソースコードのほか、ブループリントの開発、テスト、公開に必要なツールやリソースが含まれています。リリースワークフローが生成され、ブループリントが自動的にスペースに公開されました。
9. ブループリントとブループリントプロジェクトが作成されたので、次のステップはソースコードを更新してプロジェクトを設定することです。Dev Environments を使用すると、ソースリポジトリをブラウザで直接開いて編集できます。

   ナビゲーションペインで [コード] を選択し、次に [開発環境] を選択します。
10. 「開発環境を作成」を選択し、「AWS Cloud9 (ブラウザで)」を選択します。
11. デフォルト設定のまま、「作成」を選択します。
12. AWS Cloud9 ターミナルで、以下のコマンドを実行してブループリントプロジェクトディレクトリに移動します。

    ```
    cd react-app-blueprint
    ```
13. ブループリントが作成されると、`static-assets`フォルダが作成され、自動的に入力されます。このチュートリアルでは、デフォルトフォルダーを削除して、react app ブループリント用の新しいフォルダーを生成します。

    以下のコマンドを実行して static-assets フォルダーを削除します。

    ```
    rm -r static-assets
    ```

    AWS Cloud9 Linux ベースのプラットフォーム上に構築されています。Windows オペレーティングシステムを使用している場合は、代わりに以下のコマンドを使用できます。

```
rmdir /s /q static-assets
```

14. デフォルトフォルダーが削除されたので、以下のコマンドを実行して react-app `static-assets` ブループリント用のフォルダーを作成します。

    ```
    npx create-react-app static-assets
    ```

    プロンプトが表示されたら、Enter キーを押して続行します。y

    必要なパッケージを含む新しい react `static-assets` アプリケーションがフォルダーに作成されました。 CodeCatalyst 変更はリモートのソースリポジトリにプッシュする必要があります。

15. 最新の変更があることを確認し、以下のコマンドを実行して変更をコミットし、 CodeCatalyst ブループリントのソースリポジトリにプッシュします。

    ```
    git pull
    ```

    ```
    git add .
    ```

    ```
    git commit -m "Add React app to static-assets"
    ```

    ```
    git push
    ```

ブループリントのソースリポジトリに変更がプッシュされると、リリースワークフローが自動的に開始されます。このワークフローは、ブループリントのバージョンを増やし、ブループリントを構築して、スペースに公開します。次のステップでは、リリースワークフローの実行に移動して、実行状況を確認します。

ステップ 2: リリースワークフローを表示する

1. CodeCatalyst コンソールのナビゲーションペインで [CI/CD] を選択し、次に [ワークフロー] を選択します。
2. ブループリントとリリースのワークフローを選択します。
3. このワークフローには、ブループリントを構築して公開するアクションがあることがわかります。

4. [最新の実行] で [ワークフローの実行] リンクを選択すると、加えたコード変更による実行が表示されます。
5. 実行が完了すると、新しいブループリントバージョンが公開されます。公開されたブループリントバージョンはスペースの設定で確認できますが、スペースのブループリントカタログに追加されるまでプロジェクトでは使用できません。次のステップでは、ブループリントをカタログに追加します。

ステップ 3: ブループリントをカタログに追加する

ブループリントをスペースのブループリントカタログに追加すると、そのブループリントをスペース内のすべてのプロジェクトで使用できるようになります。スペースメンバーは、ブループリントを使用して新しいプロジェクトを作成したり、既存のプロジェクトに適用したりできます。

1. CodeCatalyst コンソールで、スペースに戻ります。
2. [設定] を選択し、[ブループリント] を選択します。
3. を選択しreact-app-blueprint、[カタログに追加] を選択します。
4. [保存] を選択します。

ステップ 4: ブループリントを使用してプロジェクトを作成する

ブループリントがカタログに追加されたので、プロジェクトで使用できるようになりました。このステップでは、作成したブループリントを使用してプロジェクトを作成します。後のステップでは、ブループリントの新しいバージョンを更新して公開することで、このプロジェクトを更新します。

1. [プロジェクト] タブを選択し、[プロジェクトの作成] を選択します。
2. [Space ブループリント] を選択し、を選択しますreact-app-blueprint。

   > **Note**
   >
   > ブループリントを選択すると、ブループリントのファイルの内容が表示されます。`README.md`

3. [次へ] をクリックします。
4.

   > **Note**
   >
   > このプロジェクト作成ウィザードの内容はブループリントで設定できます。

ブループリントユーザーとしてプロジェクト名を入力します。このチュートリアルでは、react-app-project と入力します。詳細については、「プロジェクト要件を満たすためのカスタムブループリントの開発」を参照してください。

次に、ブループリントを更新し、新しいバージョンをカタログに追加します。このカタログを使用して、このプロジェクトの更新を行います。

ステップ 5: ブループリントを更新する

ブループリントを使用して新しいプロジェクトを作成したり、既存のプロジェクトに適用したりした後も、ブループリントの作成者として引き続き更新を行うことができます。このステップでは、ブループリントに変更を加え、新しいバージョンをスペースに自動的に公開します。その後、新しいバージョンをカタログバージョンとして追加できます。

1. react-app-blueprintで作成したプロジェクトに移動しますチュートリアル:React アプリケーションの作成と更新。
2. チュートリアル:React アプリケーションの作成と更新で作成した開発環境を開きます。
   a. ナビゲーションペインで [コード] を選択し、次に [開発環境] を選択します。
   b. テーブルから Dev Environment を探し、「Open in AWS Cloud9 (ブラウザで)」を選択します。
3. ブループリントリリースのワークフローを実行すると、ファイルが更新されてブループリントのバージョンが増えました。package.jsonターミナルで以下のコマンドを実行して、その変更を取り込みます。 AWS Cloud9

```
git pull
```

4. static-assets以下のコマンドを実行してフォルダーに移動します。

```
cd /projects/react-app-blueprint/static-assets
```

5. 以下のコマンドを実行して、hello-world.txtstatic-assetsフォルダー内にファイルを作成します。

```
touch hello-world.txt
```

AWS Cloud9 Linux ベースのプラットフォーム上に構築されています。Windows オペレーティングシステムを使用している場合は、代わりに以下のコマンドを使用できます。

```
echo > hello-world.text
```

6. 左側のナビゲーションで、hello-world.txtファイルをダブルクリックして エディターで開き、次の内容を追加します。

```
Hello, world!
```

ファイルを保存します。

7. 最新の変更があることを確認し、以下のコマンドを実行して変更をコミットし、 CodeCatalyst ブループリントのソースリポジトリにプッシュします。

```
git pull
```

```
git add .
```

```
git commit -m "prettier setup"
```

```
git push
```

変更をプッシュするとリリースワークフローが開始され、新しいバージョンのブループリントがスペースに自動的に公開されます。

ステップ 6: ブループリントの公開済みカタログバージョンを新しいバージョンに更新する

ブループリントを使用して新しいプロジェクトを作成したり、既存のプロジェクトに適用した後でも、ブループリントの作成者としてブループリントを更新できます。このステップでは、ブループリントを変更し、ブループリントのカタログバージョンを変更します。

1. CodeCatalyst コンソールで、スペースに戻ります。
2. [設定] を選択し、[ブループリント] を選択します。
3. を選択しreact-app-blueprint、[カタログバージョンの管理] を選択します。

4. 新しいバージョンを選択し、[保存] を選択します。

ステップ 7: 新しいブループリントバージョンでプロジェクトを更新する

これで、スペースのブループリントカタログに新しいバージョンが追加されました。ブループリントユーザーは、で作成したプロジェクトのバージョンを更新できます。ステップ 4: ブループリントを使用してプロジェクトを作成するこれにより、ベストプラクティスを満たすために必要な最新の変更や修正を確実に入手できます。

1. CodeCatalyst コンソールで、react-app-projectで作成したプロジェクトに移動しますステップ 4: ブループリントを使用してプロジェクトを作成する。
2. ナビゲーションペインで [Blueprints] (ブループリント) を選択します。
3. 情報ボックスで [ブループリントを更新] を選択します。
4. 右側のコード変更パネルには、`hello-world.txtpackage.json`と更新が表示されます。
5. 「アップデートを適用」を選択します。

[Apply update] を選択すると、更新されたブループリントバージョンからの変更を含むプルリクエストがプロジェクトに作成されます。プロジェクトを更新するには、プルリクエストをマージする必要があります。詳細については、「プルリクエストの確認」および「プルリクエストのマージ」を参照してください。

1. ブループリントテーブルで、ブループリントを探します。「ステータス」列で「保留中のプルリクエスト」を選択し、オープン中のプルリクエストへのリンクを選択します。
2. プルリクエストを確認し、[マージ] を選択します。
3. [早送りマージ] を選択してデフォルト値をそのまま使用し、[マージ] を選択します。

ステップ 8: プロジェクトの変更を表示する

ブループリントへの変更は、ステップ 7: 新しいブループリントバージョンでプロジェクトを更新するあとからプロジェクトで利用できるようになります。ブループリントユーザーは、ソースリポジトリ内の変更を確認できます。

1. ナビゲーションペインで [Source repositories] を選択し、プロジェクトの作成時に作成されたソースリポジトリの名前を選択します。
2. [ファイル] には、`hello-world.txt`ステップ 5: ブループリントを更新するで作成されたファイルが表示されます。

3. を選択するとhello-world.txt、ファイルの内容が表示されます。

ライフサイクル管理により、ブループリント作成者は特定のブループリントを含むすべてのプロジェクトのソフトウェア開発ライフサイクルを一元管理できます。このチュートリアルで説明したように、ブループリントに更新をプッシュして、そのブループリントを使用して新しいプロジェクトを作成したり、既存のプロジェクトに適用したりしたプロジェクトに組み込むことができます。詳細については、「ブループリントの作成者としてライフサイクル管理を使用する」を参照してください。

ブループリントの作成者としてライフサイクル管理を使用する

ライフサイクル管理を使用すると、1 つの一般的なベストプラクティスのソースから多数のプロジェクトを同期させることができます。これにより、修正の伝播と、ソフトウェア開発ライフサイクル全体にわたる任意の数のプロジェクトのメンテナンスがスケールされます。ライフサイクル管理は、内部キャンペーン、セキュリティ修正、監査、ランタイムアップグレード、ベストプラクティスの変更、その他のメンテナンスプラクティスを効率化します。これらの標準は 1 か所で定義され、新しい標準が公開されると自動的に最新の状態に保たれるためです。

ブループリントの新しいバージョンが公開されると、そのブループリントを含むすべてのプロジェクトが最新バージョンに更新するように求められます。設計図の作成者は、各プロジェクトに含まれる特定の設計図のバージョンをコンプライアンスの目的で確認することもできます。既存のソースリポジトリで競合が発生すると、ライフサイクル管理によってプルリクエストが作成されます。開発環境などの他のすべてのリソースについては、すべてのライフサイクル管理の更新によって、厳密に新しいリソースが作成されます。ユーザーは、これらのプルリクエストをマージするかどうかは自由に選択できます。保留中のプルリクエストがマージされると、プロジェクトで使用されるオプションを含むブループリントのバージョンが更新されます。ブループリントユーザーとしてライフサイクル管理を使用する方法については、既存のプロジェクトでのライフサイクル管理の使用「」および「」を参照してくださいプロジェクト内の複数のブループリントでのライフサイクル管理の使用。

トピック

- バンドル出力とマージ競合のライフサイクル管理のテスト
- マージ戦略を使用してバンドルを生成し、ファイルを指定する
- プロジェクトの詳細のコンテキストオブジェクトへのアクセス

バンドル出力とマージ競合のライフサイクル管理のテスト

ブループリントのライフサイクル管理をローカルでテストし、競合の解決をマージできます。ライフサイクル更新のさまざまなフェーズを表す一連のバンドルが synth/ ディレクトリの下に生成

されます。ライフサイクル管理をテストするには、設計図で次の yarn コマンドを実行しますyarn blueprint: resynth。再合成とバンドルの詳細については、「」および再合成「」を参照してください再合成によるファイルの生成。

マージ戦略を使用してバンドルを生成し、ファイルを指定する

トピック

- 再合成によるファイルの生成
- マージ戦略の使用
- ライフサイクル管理更新用のファイルの指定
- マージ戦略の記述

再合成によるファイルの生成

再合成では、ブループリントによって生成されたソースコードを、同じブループリントによって以前に生成されたソースコードとマージできるため、ブループリントへの変更を既存のプロジェクトに伝達できます。マージは、設計図出力バンドル全体で resynth( ) 関数から実行されます。再合成では、まず設計図とプロジェクトの状態のさまざまな側面を表す 3 つのバンドルが生成されます。yarn blueprint:resynth コマンドを使用してローカルで手動で実行できます。これにより、バンドルが存在しない場合はバンドルが作成されます。バンドルを手動で操作すると、再合成動作をモックしてローカルでテストできます。デフォルトでは、バンドルのその部分のみが通常出典管理下にあるsrc/*ため、ブループリントは のリポジトリ間でのみ再合成を実行します。

- existing-bundle - このバンドルは、既存のプロジェクトの状態を表します。これは、合成コンピューティングによって人工的に構築され、デプロイ先のプロジェクトの内容 (ある場合) に関する設計図コンテキストを提供します。再合成をローカルで実行するときに、この場所に既に存在するものがある場合、リセットされ、モックとして認識されます。それ以外の場合は、 の内容に設定されますancestor-bundle。
- ancestor-bundle - これは、以前のオプションやバージョンで合成された場合に設計図出力を表すバンドルです。このブループリントをプロジェクトに初めて追加する場合、祖先が存在しないため、 と同じ内容に設定されますexisting-bundle。ローカルでは、このバンドルがこの場所に既に存在する場合、モックとして扱われます。
- proposed-bundle - これは、いくつかの新しいオプションやバージョンで合成された場合に、ブループリントをモックするバンドルです。これは、synth( )関数によって生成されるバンドルと同じです。ローカルでは、このバンドルは常に上書きされます。

各バンドルは再合成フェーズ中に作成され、 のブループリントクラスからアクセスできますthis.context.resynthesisPhase。

- resolved-bundle - これは最終バンドルであり、パッケージ化されて CodeCatalyst プロジェクトにデプロイされる内容を表します。デプロイメカニズムに送信されるファイルと差分を表示できます。これは、他の 3 つのバンドル間のマージを解決するresynth()関数の出力です。

3 方向マージは、 ancestor-bundle と の違いを に適用proposed-bundleし、それを に適用existing-bundleして を生成することによって適用されますresolved-bundle。すべてのマージ戦略は、ファイルを に解決しますresolved-bundle。再合成は、 中にブループリントのマージ戦略を使用してこれらのバンドルの到達範囲を解決resynth()し、結果から解決されたバンドルを生成します。

マージ戦略の使用

ブループリントライブラリが提供するマージ戦略を使用できます。これらの戦略は、 再合成によるファイルの生成セクションで説明されているファイルの出力と競合を解決する方法を提供します。

- alwaysUpdate - 常に提案されたファイルに解決される戦略。
- neverUpdate - 常に既存のファイルに解決される戦略。
- onlyAdd - 既存のファイルがまだ存在しない場合に、提案されたファイルに解決される戦略。それ以外の場合、 は既存のファイルに解決されます。
- threeWayMerge - 既存の祖先ファイル、提案された祖先ファイル、一般的な祖先ファイルの間で 3 方向のマージを実行する戦略。ファイルをクリーンにマージできない場合、解決されたファイルに競合マーカーが含まれている可能性があります。戦略が意味のある出力を生成するには、提供されたファイルの内容が UTF-8 でエンコードされている必要があります。戦略は、入力ファイルがバイナリであるかどうかを検出しようとします。戦略がバイナリファイル内のマージ競合を検出すると、常に提案されたファイルを返します。
- preferProposed - 既存の祖先ファイル、提案された祖先ファイル、一般的な祖先ファイル間で 3 方向のマージを実行する戦略。この戦略では、提案されたファイルの各競合の側を選択して競合を解決します。
- preferExisting - 既存の祖先ファイル、提案された祖先ファイル、一般的な祖先ファイルの間で 3 方向のマージを実行する戦略。この戦略では、各競合の既存のファイルの 側を選択して競合を解決します。

マージ戦略のソースコードを表示するには、「オープンソース GitHub リポジトリ」を参照してください。

ライフサイクル管理更新用のファイルの指定

再同期中、ブループリントは変更を既存のソースリポジトリにマージする方法を制御します。ただし、設計図のすべてのファイルに更新をプッシュしたくない場合があります。例えば、CSS スタイルシートなどのサンプルコードはプロジェクト固有であることが意図されています。別の戦略を指定しない場合、3 方向マージ戦略がデフォルトのオプションです。設計図では、リポジトリコンストラクト自体でマージ戦略を指定することで、所有するファイルと所有しないファイルを指定できます。ブループリントはマージ戦略を更新でき、最新の戦略は再合成中に使用できます。

```
const sourceRepo = new SourceRepository(this, {
      title: 'my-repo',
    });
    sourceRepo.setResynthStrategies([
      {
        identifier: 'dont-override-sample-code',
        description: 'This strategy is applied accross all sample code. The blueprint
 will create sample code, but skip attempting to update it.',
        strategy: MergeStrategies.neverUpdate,
        globs: [
          '**/src/**',
          '**/css/**',
        ],
      },
    ]);
```

複数のマージ戦略を指定でき、最後の戦略が優先されます。カバーされていないファイルは、デフォルトで Git に似ています three-way-merge 。MergeStrategies コンストラクトを通じて提供されるマージ戦略はいくつかありますが、独自のマージ戦略を記述することはできます。提供される戦略は、git マージ戦略ドライバーに準拠しています。

マージ戦略の記述

提供されているビルドマージ戦略の 1 つを使用することに加えて、独自の戦略を作成することもできます。戦略は、標準戦略インターフェイスに従う必要があります。existing-bundle、、proposed-bundleおよび からファイルのバージョンを取得しancestor-bundle、それらを 1 つの解決済みファイルにマージする戦略関数を記述する必要があります。例:

```
type StrategyFunction = (
```

```
   /**
   * file from the ancestor bundle (if it exists)
   */
    commonAncestorFile: ContextFile | undefined,
   /**
   * file from the existing bundle (if it exists)
   */
    existingFile: ContextFile | undefined,
   /**
   * file from the proposed bundle (if it exists)
   */
    proposedFile: ContextFile | undefined,
    options?: {})
    /**
    * Return: file you'd like in the resolved bundle
    * passing undefined will delete the file from the resolved bundle
    */
=> ContextFile | undefined;
```

ファイルが存在しない (未定義) 場合、そのファイルパスはその特定の口ケーションバンドルには存在しません。

例:

```
strategies: [
          {
            identifier: 'dont-override-sample-code',
            description: 'This strategy is applied across all sample code. The
 blueprint will create sample code, but skip attempting to update it.',
            strategy: (ancestor, existing, proposed) => {
                const resolvedfile = ...
                ...
                // do something
                ...
                return resolvedfile
            },
            globs: [
              '**/src/**',
              '**/css/**',
            ],
          },
        ],
```

プロジェクトの詳細のコンテキストオブジェクトへのアクセス

設計図の作成者は、合成中に設計図のプロジェクトからコンテキストにアクセスして、スペースやプロジェクト名、プロジェクトのソースリポジトリ内の既存のファイルなどの情報を取得できます。設計図が生成している再合成のフェーズなどの詳細を取得することもできます。例えば、コンテキストにアクセスして、祖先バンドルまたは提案されたバンドルを生成するために再合成するかどうかを確認できます。その後、既存のコードコンテキストを使用してリポジトリ内のコードを変換できます。例えば、独自の再合成戦略を記述して、特定のコード標準を設定できます。戦略は、小さなブループリントの blueprint.ts ファイルに追加することも、戦略用に別のファイルを作成することもできます。

次の例は、プロジェクトのコンテキストでファイルを検索し、ワークフロービルダーを設定し、特定のファイルに対してブループリントで提供される再合成戦略を設定する方法を示しています。

```
const contextFiles = this.context.project.src.findAll({
      fileGlobs: ['**/package.json'],
    });

    // const workflows = this.context.project.src.findAll({
    //   fileGlobs: ['**/.codecatalyst/**.yaml'],
    // });

    const security = new WorkflowBuilder(this, {
      Name: 'security-workflow',
    });
    new Workflow(this, repo, security.getDefinition());
    repo.setResynthStrategies([
      {
        identifier: 'force-security',
        globs: ['**/.codecatalyst/security-workflow.yaml'],
        strategy: MergeStrategies.alwaysUpdate,
      },
    ]);


    for (const contextFile of contextFiles) {
      const packageObject = JSON.parse(contextFile.buffer.toString());
      new SourceFile(internalRepo, contextFile.path, JSON.stringify({
        ...packageObject,
      }, null, 2));
    }
  }
```

プロジェクト要件を満たすためのカスタムブループリントの開発

カスタムブループリントを公開する前に、特定の要件を満たすようにブループリントを開発できます。プレビュー時にプロジェクトを作成することで、カスタムブループリントを開発し、ブループリントをテストできます。カスタムブループリントを開発して、特定のソースコード、アカウント接続、ワークフロー、問題、または で作成できるその他のコンポーネントなどのプロジェクトコンポーネントを含めることができます CodeCatalyst。

> **Important**
>
> 外部ソースからのブループリントパッケージを使用する場合は、それらのパッケージに伴うリスクを考慮してください。スペースに追加するカスタムブループリントとそれらが生成するコードは、お客様の責任となります。

カスタムブループリントを開発または更新するには

1. 開発環境を再開します。詳細については、「開発環境の再開」を参照してください。

   開発環境がない場合は、まず開発環境を作成する必要があります。詳細については、「開発環境の作成」を参照してください。
2. 開発環境で作業ターミナルを開きます。
3. ブループリントの作成時にリリースワークフローをオプトインすると、最新のブループリントバージョンが自動的に公開されます。変更をプルして、package.jsonファイルに増分バージョンがあることを確認します。以下のコマンドを使用します。

   ```
   git pull
   ```
4. src/blueprint.ts ファイルで、カスタムブループリントのオプションを編集します。Options インターフェイスは CodeCatalyst ウィザードによって動的に解釈され、選択ユーザーインターフェイス (UI) が生成されます。コンポーネントとサポートされているタグを追加することで、カスタムブループリントを開発できます。詳細については、「フロントエンドウィザードを使用した設計図機能の変更」、「設計図への環境コンポーネントの追加」、「設計図へのリージョンコンポーネントの追加」、「設計図へのリポジトリとソースコードコンポーネントの追加」、「設計図へのワークフローコンポーネントの追加」、「開発環境コンポーネントを設計図に追加する」を参照してください。

カスタムブループリントの開発時に、追加のサポートのためにブループリント SDK とサンプルブループリントを表示することもできます。詳細については、「オープンソースリポジトリ GitHub 」を参照してください。

カスタムブループリントは、合成が成功した結果、プレビューバンドルを提供します。プロジェクトバンドルは、プロジェクト内のソースコード、設定、リソースを表し、 CodeCatalyst デプロイ API オペレーションによってプロジェクトにデプロイするために使用されます。カスタムブループリントの開発を続行する場合は、ブループリント合成プロセスを再実行します。詳細については、「カスタムブループリントの概念」を参照してください。

フロントエンドウィザードを使用した設計図機能の変更

のブループリント選択ウィザード CodeCatalyst は、 `blueprint.ts` ファイルの `Options` インターフェイスによって自動的に生成されます。フロントエンドウィザードは、JSDOC スタイルのコメントとタグ`Options`を使用して、ブループリントの の変更と機能をサポートします。 https://jsdoc.app/about-getting-started.htmlJSDOC スタイルのコメントとタグを使用してタスクを実行できます。例えば、オプションの上に表示されるテキストの選択、入力検証などの機能の有効化、またはオプションを折りたたみ可能にすることができます。ウィザードは、インターフェイスから TypeScript タイプから生成された抽象構文ツリー (AST) を解釈することで機能します`Options`。ウィザードは、可能な限り最適なタイプに自動的に設定します。すべてのタイプがサポートされているわけではありません。サポートされている他のタイプには、リージョンセレクタと環境セレクタがあります。

以下は、設計図の で JSDOC コメントとタグを使用するウィザードの例です`Options`。

```
export interface Options {
  /**
   * What do you want to call your new blueprint?
   * @validationRegex /^[a-zA-Z0-9_]+$/
   * @validationMessage Must contain only upper and lowercase letters, numbers and
 underscores
   */
  blueprintName: string;

  /**
   * Add a description for your new blueprint.
   */
   description?: string;
```

```
   /**
    * Tags for your Blueprint:
    * @collapsed true
    */
  tags?: string[];
}
```

Options インターフェイスの各オプションの表示名は、camelCaseデフォルトで に表示されます。JSDOC スタイルのコメントのプレーンテキストは、ウィザードの オプションの上にテキストとして表示されます。

トピック

- サポートされているタグ
- サポートされている TypeScript タイプ
- 合成中にユーザーと通信する

サポートされているタグ

次の JSDOC タグは、フロントエンドウィザードのカスタムブループリントの Optionsでサポートされています。

@inlinePolicy ./path/to/policy/file.json

- 必須 - オプションをタイプ にする必要がありますRole。
- 使用状況 - ロールに必要なインラインポリシーを通信できます。policy.json パスはソースコードの下に存在することが想定されます。ロールのカスタムポリシーが必要な場合は、このタグを使用します。
- 依存関係 - blueprint-cli 0.1.12以上
- 例 - @inlinePolicy ./deployment-policy.json

```
environment: EnvironmentDefinition{
    awsAccountConnection: AccountConnection{
      /**
       * @inlinePolicy ./path/to/deployment-policy.json
       */
      cdkRole: Role[];
    };
   };
```

@trustPolicy ./path/to/policy/file.json

- 必須 - オプションをタイプ にする必要がありますRole。
- 使用状況 - ロールに必要な信頼ポリシーを通信できます。policy.json パスはソースコードの下に存在することが想定されます。ロールのカスタムポリシーが必要な場合は、このタグを使用します。
- 依存関係 - blueprint-cli 0.1.12以上
- 例 - @trustPolicy ./trust-policy.json

```
environment: EnvironmentDefinition{
    awsAccountConnection: AccountConnection{
      /**
       * @trustPolicy ./path/to/trust-policy.json
       */
      cdkRole: Role[];
    };
   };
```

@validationRegex 正規表現式

- 必須 - オプションを文字列にする必要があります。
- 使用状況 - 指定された正規表現式を使用して オプションの入力検証を実行し、 を表示します@validationMessage。
- 例 - @validationRegex /^[a-zA-Z0-9_]+$/
- レコメンデーション - で使用します@validationMessage。デフォルトでは、検証メッセージは空です。

@validationMessage 文字列

- 使用状況を確認するには、 @validationRegexまたはその他のエラーが必要です。
- 使用状況 - @validation*障害発生時の検証メッセージを表示します。
- 例 - @validationMessage Must contain only upper and lowercase letters, numbers, and underscores。
- レコメンデーション - で使用します@validationMessage。デフォルトでは、検証メッセージは空です。

@collapsed ブール値 (オプション)

- 必須 - 該当なし
- 使用状況 - サブオプションを折りたためるようにするブール値。折りたたまれたアノテーションが存在する場合、デフォルト値は true です。値を に設定すると、最初は開いている折りたたみ可能なセクション`@collapsed false`が作成されます。
- 例 - `@collapsed true`

@displayName 文字列

- 必須 - 該当なし
- Usage - オプションの表示名を変更します。表示名に camelCase 以外の形式を許可します。
- 例 - `@displayName Blueprint Name`

@displayName 文字列

- 必須 - 該当なし
- Usage - オプションの表示名を変更します。表示名に camelCase 以外の形式を許可します。
- 例 - `@displayName Blueprint Name`

@defaultEntropy 番号

- 必須 - オプションを文字列にする必要があります。
- Usage - 指定された長さのランダム化された英数字の文字列を オプションに追加します。
- 例 - `@defaultEntropy 5`

@placeholder 文字列 (オプション)

- 必須 - 該当なし
- Usage - デフォルトのテキストフィールドプレースホルダーを変更します。
- 例 - `@placeholder type project name here`

@textArea number (オプション)

- 必須 - 該当なし

- 使用法 - 文字列入力をテキストエリアコンポーネントに変換して、テキストの大きなセクションにします。数値を追加すると、行数が定義されます。デフォルトは 5 行です。
- 例 - `@textArea 10`

@hidden ブール値 (オプション)

- 必須 - 該当なし
- 使用状況 - 検証チェックが失敗しない限り、ユーザーからファイルを非表示にします。デフォルト値は true です。
- 例 - `@hidden`

@button ブール値 (オプション)

- 必須 - 該当なし
- Usage - 注釈はブールプロパティ上にある必要があります。選択すると true として合成されるボタンを追加します。トグルではありません。
- 例 - `buttonExample: boolean;`

```
/**
  * @button
  */
buttonExample: boolean;
```

@showName ブール値 (オプション)

- 必須 - 該当なし
- 使用状況 - アカウント接続タイプでのみ使用できます。非表示の名前入力を表示します。デフォルトは `default_environment` です。
- 例 - `@showName true`

```
/**
  * @showName true
  */
accountConnection: AccountConnection<{
    ...
}>;
```

@showEnvironmentType boolean (オプション)

- 必須 - 該当なし
- 使用状況 - アカウント接続タイプでのみ使用できます。非表示の環境タイプのドロップダウンメニューを表示します。すべての接続のデフォルトは ですproduction。オプションは、非本番稼働用 または本番稼働用 です。
- 例 - `@showEnvironmentType true`

```
/**
  * @showEnvironmentType true
  */
accountConnection: AccountConnection<{
    ...
}>;
```

@forceDefault ブール値 (オプション)

- 必須 - 該当なし
- 使用状況 - ユーザーが以前に使用した値の代わりに、設計図の作成者が提供したデフォルト値を使用します。
- 例 - `forceDeafultExample: any;`

```
/**
  * @forceDefault
  */
forceDeafultExample: any;
```

@requires blueprintName

- 必須 - Optionsインターフェイスに注釈を付けます
- 使用状況 - 現在のブループリントの要件としてプロジェクトblueprintNameに指定された を適用するようにユーザーに警告します。
- 例 - `@requires '@amazon-codecatalyst/blueprints.blueprint-builder'`

```
/*
 * @requires '@amazon-codecatalyst/blueprints.blueprint-builder'
 */
```

```
export interface Options extends ParentOptions {
...
```

サポートされている TypeScript タイプ

以下の TypeScript タイプは、フロントエンドウィザードのカスタムブループリントの `Options`でサポートされています。

数

- 必須 - オプションをタイプ にする必要がありますnumber。
- Usage - 数値入力フィールドを生成します。
- 例 - `age: number`

```
{
  age: number
  ...
}
```

文字列

- 必須 - オプションをタイプ にする必要がありますstring。
- Usage - 文字列入力フィールドを生成します。
- 例 - `name: string`

```
{
  age: string
  ...
}
```

文字列リスト

- 必須 - オプションをタイプ にする必要がありますboolean。
- 使用状況 - チェックボックスを生成します。
- 例 - `isProduction: boolean`

```
{
  isProduction: boolean
  ...
}
```

無線

- 必須 - オプションは 3 つ以下の文字列の和集合です。
- 使用状況 - 選択した無線を生成します。

> **Note**
>
> 4 つ以上の項目がある場合、このタイプはドロップダウンとしてレンダリングされます。

- 例 - `color: 'red' | 'blue' | 'green'`

```
{
  color: 'red' | 'blue' | 'green'
  ...
}
```

ドロップダウン

- 必須 - オプションは 4 つ以上の文字列の和集合です。
- 使用状況 - ドロップダウンを生成します。
- 例 - `runtimes: 'nodejs' | 'python' | 'java' | 'dotnetcore' | 'ruby'`

```
{
  runtimes: 'nodejs' | 'python' | 'java' | 'dotnetcore' | 'ruby'
  ...
}
```

展開可能なセクション

- 必須 - オプションをオブジェクトにします。
- 使用状況 - 拡張可能なセクションを生成します。オブジェクトのオプションは、ウィザードの展開可能なセクション内にネストされます。

- 例 -

```
{
     expandableSectionTitle: {
         nestedString: string;
         nestedNumber: number;
     }
}
```

タプル

- 必須 - オプションのタイプは ですTuple。
- Usage - キーと値の有料入力を生成します。
- 例 - `tuple: Tuple[string, string]>`

```
{
    tuple: Tuple[string, string]>;
    ...
}
```

タプルリスト

- 必須 - オプションを 型の配列にする必要がありますTuple。
- Usage - タプルリスト入力を生成します。
- 例 - `tupleList: Tuple[string, string]>[]`

```
{
  tupleList: Tuple[string, string]>[];
  ...
}
```

Selector

- 必須 - オプションのタイプは ですSelector。
- 使用状況 - プロジェクトに適用されるソースリポジトリまたはブループリントのドロップダウンを生成します。

- 例 - `sourceRepo: Selector<SourceRepository>`

```
{
    sourceRepo: Selector<SourceRepository>;
    sourceRepoOrAdd: Selector<SourceRepository | string>;
    blueprintInstantiation: Selector<BlueprintInstantiation>;
  ...
}
```

複数選択

- 必須 - オプションのタイプは ですSelector。
- 使用状況 - 複数選択入力を生成します。
- 例 - `multiselect: MultiSelect['A' | 'B' | 'C' | 'D' | 'E']>`

```
{
  multiselect: MultiSelect['A' | 'B' | 'C' | 'D' | 'E']>;
  ...
}
```

合成中にユーザーと通信する

設計図の作成者は、検証メッセージだけでなく、ユーザーと通信することもできます。例えば、スペースメンバーが、明確でないブループリントを生成するオプションの組み合わせを表示する場合があります。カスタムブループリントは、合成を呼び出すことでエラーメッセージをユーザーに返す機能をサポートしています。基本ブループリントは、明確なエラーメッセージを期待するthrowSynthesisError(...)関数を実装します。以下を使用してメッセージを呼び出すことができます。

```
//blueprint.ts
this.throwSynthesisError({
   name: BlueprintSynthesisErrorTypes.BlueprintSynthesisError,
   message: 'hello from the blueprint! This is a custom error communicated to the
 user.'
})
```

設計図への環境コンポーネントの追加

カスタムブループリントウィザードは、ウィザードによって公開されるOptionsインターフェイスから動的に生成されます。ブループリントは、公開されたタイプからのユーザーインターフェイス (UI) コンポーネントの生成をサポートします。

Amazon CodeCatalyst ブループリント環境コンポーネントをインポートするには

`blueprint.ts` ファイルで、以下を追加します。

```
import {...} from '@amazon-codecatalyst/codecatalyst-environments'
```

トピック

- 開発環境の作成
- モックインターフェイスの例

開発環境の作成

次の例は、アプリケーションをクラウドにデプロイする方法を示しています。

```
export interface Options extends ParentOptions {
        ...
        myNewEnvironment:  EnvironmentDefinition{
            thisIsMyFirstAccountConnection: AccountConnection{
                thisIsARole: Role['lambda', 's3', 'dynamo'];
             };
        };
    }
```

インターフェイスは、単一のアカウント接続 () を持つ新しい環境 (myNewEnvironment) を要求する UI コンポーネントを生成しますthisIsMyFirstAccountConnection。 アカウント接続 (thisIsARole) のロールは、最低限必要なロール機能['lambda', 's3', 'dynamo']としてでも生成されます。すべてのユーザーがアカウント接続を持っているわけではないため、ユーザーがアカウントに接続しない場合や、アカウントをロールに接続しない場合にチェックする必要があります。ロールには の注釈を付けることもできます@inlinePolicies。詳細については、「@inlinePolicy ./path/to/policy/file.json」を参照してください。

環境コンポーネントには nameと が必要ですenvironmentType。次のコードは、最低限必要なデフォルトの形状です。

```
{
  ...
  "myNewEnvironment": {
    "name": "myProductionEnvironment",
    "environmentType": "PRODUCTION"
  },
}
```

UI コンポーネントは、さまざまなフィールドの入力を求めます。フィールドに入力すると、設計図が完全に展開された形状になります。defaults.json テストと開発の目的で、 ファイルの完全なモックを含めると便利です。

モックインターフェイスの例

シンプルなモックインターフェイス

```
{
    ...
    "thisIsMyEnvironment": {
        "name": "myProductionEnvironment",
        "environmentType": "PRODUCTION",
        "thisIsMySecondAccountConnection": {
            "id": "12345678910",
            "name": "my-account-connection-name",
            "secondAdminRole": {
                "arn": "arn:aws:iam::12345678910:role/ConnectedQuokkaRole",
                "name": "ConnectedQuokkaRole",
                "capabilities": [
                    "lambda",
                    "s3",
                    "dynamo"
                ]
            }
        }
    }
}
```

複雑なモックインターフェイス

```
export interface Options extends ParentOptions {
  /**
   * The name of an environment
```

```
   * @displayName This is a Environment Name
   * @collapsed
   */
  thisIsMyEnvironment: EnvironmentDefinition{
    /**
     * comments about the account that is being deployed into
     * @displayName This account connection has an overriden name
     * @collapsed
     */
    thisIsMyFirstAccountConnection: AccountConnection{
      /**
       * Blah blah some information about the role that I expect
       * e.g. here's a copy-pastable policy: [to a link]
       * @displayName This role has an overriden name
       */
      adminRole: Role['admin', 'lambda', 's3', 'cloudfront'];
      /**
       * Blah blah some information about the second role that I expect
       * e.g. here's a copy-pastable policy: [to a link]
       */
      lambdaRole: Role['lambda', 's3'];
    };
    /**
     * comments about the account that is being deployed into
     */
    thisIsMySecondAccountConnection: AccountConnection{
      /**
         * Blah blah some information about the role that I expect
         * e.g. here's a copy-pastable policy: [to a link]
         */
      secondAdminRole: Role['admin', 'lambda', 's3', 'cloudfront'];
      /**
         * Blah blah some information about the second role that I expect
         * e.g. here's a copy-pastable policy: [to a link]
         */
      secondLambdaRole: Role['lambda', 's3'];
    };
  };
}
```

完全なモックインターフェイス

```
{
```

```
  ...
  "thisIsMyEnvironment": {
    "name": "my-production-environment",
    "environmentType": "PRODUCTION",
    "thisIsMySecondAccountConnection": {
      "id": "12345678910",
      "name": "my-connected-account",
      "secondAdminRole": {
        "name": "LambdaQuokkaRole",
        "arn": "arn:aws:iam::12345678910:role/LambdaQuokkaRole",
        "capabilities": [
          "admin",
          "lambda",
          "s3",
          "cloudfront"
        ]
      },
      "secondLambdaRole": {
        "name": "LambdaQuokkaRole",
        "arn": "arn:aws:iam::12345678910:role/LambdaQuokkaRole",
        "capabilities": [
          "lambda",
          "s3"
        ]
      }
    },
    "thisIsMyFirstAccountConnection": {
      "id": "12345678910",
      "name": "my-connected-account",
      "adminRole": {
        "name": "LambdaQuokkaRole",
        "arn": "arn:aws:iam::12345678910:role/LambdaQuokkaRole",
        "capabilities": [
          "admin",
          "lambda",
          "s3",
          "cloudfront"
        ]
      },
      "lambdaRole": {
        "name": "LambdaQuokkaRole",
        "arn": "arn:aws:iam::12345678910:role/LambdaQuokkaRole",
        "capabilities": [
          "lambda",
```

```
          "s3"
        ]
      }
    }
  },
}
```

設計図へのシークレットコンポーネントの追加

シークレットは、ワークフローで参照できる機密データを保存 CodeCatalyst するために で使用できます。シークレットをカスタムブループリントに追加し、ワークフローで参照できます。詳細については、「ワークフローでのシークレットの設定と使用」を参照してください。

Amazon CodeCatalyst ブループリントのリージョンタイプをインポートするには

`blueprint.ts` ファイルで、以下を追加します。

```
import { Secret, SecretDefinition } from '@amazon-codecatalyst/blueprint-
component.secrets'
```

トピック

- シークレットの作成
- ワークフローでシークレットを参照する

シークレットの作成

次の例では、シークレット値とオプションの説明を入力するようにユーザーに求める UI コンポーネントを作成します。

```
export interface Options extends ParentOptions {
    ...
    mySecret: SecretDefinition;
}


export class Blueprint extends ParentBlueprint {
  constructor(options_: Options) {
    new Secret(this, options.secret);
}
```

シークレットコンポーネントには が必要ですname。次のコードは、最低限必要なデフォルトの形状です。

```
{
    ...
    "secret": {
        "name": "secretName"
    },

}
```

ワークフローでシークレットを参照する

次の設計図の例では、シークレットと、シークレット値を参照するワークフローを作成します。詳細については、「ワークフローでシークレットを参照する」を参照してください。

```
export interface Options extends ParentOptions {
    ...
/**
*
* @validationRegex /^\w+$/
*/
  username: string;


  password: SecretDefinition;
}


export class Blueprint extends ParentBlueprint {
  constructor(options_: Options) {
    const password = new Secret(this, options_.password);

    const workflowBuilder = new WorkflowBuilder(this, {
      Name: 'my_workflow',
    });


    workflowBuilder.addBuildAction({
      actionName: 'download_files',
      input: {
        Sources: ['WorkflowSource'],
      },
```

```
      output: {
        Artifacts: [{ Name: 'download', Files: ['file1'] }],
      },
      steps: [
        `curl -u ${options_.username}:${password.reference} https://example.com`,
      ],
    });

    new Workflow(
      this,
      repo,
      workflowBuilder.getDefinition(),
    );

}
```

でのシークレットの使用の詳細については CodeCatalyst、「」を参照してくださいワークフローでのシークレットの設定と使用。

設計図へのリージョンコンポーネントの追加

リージョンタイプをカスタムブループリントのOptionsインターフェイスに追加して、1 つ以上の AWS gions を入力できるブループリントウィザードでコンポーネントを生成できます。Gion タイプは、 blueprint.ts ファイルのベースブループリントからインポートできます。詳細については、「AWS リージョン」を参照してください。

Amazon CodeCatalyst ブループリントのリージョンタイプをインポートするには

blueprint.ts ファイルで、以下を追加します。

```
import { Region } from '@amazon-codecatalyst/blueprints.blueprint'
```

region type パラメータは、選択する AWS リージョンコードの配列です。または、 *を使用して、サポートされているすべての AWS リージョンを含めることができます。

トピック

- 注釈
- リージョンコンポーネントの例

注釈

JSDoc タグをOptionsインターフェイスの各フィールドに追加して、ウィザードでのフィールドの表示方法と動作をカスタマイズできます。リージョンタイプでは、次のタグがサポートされています。

- @displayName 注釈を使用して、ウィザードでフィールドのラベルを変更できます。

  例: `@displayName AWS Region`
- @placeholder 注釈を使用して、コンポーネントのプレースホルダーの選択/複数選択を変更できます。

  例: `@placeholder Choose AWS Region`

リージョンコンポーネントの例

指定したリストからリージョンを選択する

```
export interface Options extends ParentOptions {
    ...
  /**
   * @displayName Region
   */
  region: Region<['us-east-1', 'us-east-2', 'us-west-1', 'us-west-2']>;
}
```

指定したリストから 1 つ以上のリージョンを選択する

```
export interface Options extends ParentOptions {
    ...
  /**
   * @displayName Regions
   */
  multiRegion: Region<['us-east-1', 'us-east-2', 'us-west-1', 'us-west-2']>[];
}
```

AWS egion を 1 つ選択する

```
export interface Options extends ParentOptions {
    ...
```

```
  /**
   * @displayName Region
   */
  region: Region<['*']>;
}
```

指定したリストから 1 つ以上のリージョンを選択する

```
export interface Options extends ParentOptions {
    ...
  /**
   * @displayName Regions
   */
  multiRegion: Region<['us-east-1', 'us-east-2', 'us-west-1', 'us-west-2']>[];
}
```

設計図へのリポジトリとソースコードコンポーネントの追加

リポジトリは、Amazon がコードを保存 CodeCatalyst するために使用します。リポジトリは名前を入力として受け取ります。ほとんどのコンポーネントは、ソースコードファイル、ワークフロー、マネージド開発環境 (MDE) などの他のコンポーネントなどのリポジトリに保存されます。ソースリポジトリコンポーネントは、ファイルと静的アセットの管理に使用されるコンポーネントもエクスポートします。リポジトリには名前の制約があります。詳細については、「でソースリポジトリを使用してコードを保存し、共同作業する CodeCatalyst」を参照してください。

```
const repository = new SourceRepository(this, {
  title: 'my-new-repository-title',
});
```

Amazon CodeCatalyst ブループリントリポジトリとソースコードコンポーネントをインポートするには

`blueprint.ts` ファイルで、以下を追加します。

```
import {...} from '@caws-blueprint-component/caws-source-repositories'
```

トピック

- ファイルの追加
- 汎用ファイルの追加

- ファイルのコピー
- 複数のファイルをターゲットにする
- 新しいリポジトリの作成とファイルの追加

ファイルの追加

SourceFile コンストラクトを使用して、テキストファイルをリポジトリに書き込むことができます。オペレーションは最も一般的なユースケースの 1 つであり、リポジトリ、ファイルパス、テキストコンテンツを使用します。ファイルパスがリポジトリ内に存在しない場合、コンポーネントは必要なフォルダをすべて作成します。

```
new SourceFile(repository, `path/to/my/file/in/repo/file.txt`, 'my file contents');
```

> **Note**
>
> 同じリポジトリ内の同じ場所に 2 つのファイルを書き込むと、最新の実装によって以前のファイルが上書きされます。この機能を使用して生成されたコードをレイヤー化できます。カスタムブループリントが生成したコードを拡張する場合に特に便利です。

汎用ファイルの追加

任意のビットをリポジトリに書き込むことができます。バッファから読み取り、 Fileコンストラクトを使用できます。

```
new File(repository, `path/to/my/file/in/repo/file.img`, new Buffer(...));

new File(repository, `path/to/my/file/in/repo/new-img.img`, new StaticAsset('path/to/
image.png').content());
```

ファイルのコピー

生成されたコードの使用を開始するには、スターターコードをコピーして貼り付け、そのベース上にさらにコードを生成します。ディレクトリ内にコードを配置しstatic-assets、そのコードをStaticAssetコンストラクトでターゲットにします。この場合のパスは常に static-assets ディレクトリのルートから始まります。

```
const starterCode = new StaticAsset('path/to/file/file.txt')
```

```
const starterCodeText = new StaticAsset('path/to/file/file.txt').toString()
const starterCodeRawContent = new StaticAsset('path/to/image/hello.png').content()

const starterCodePath = new StaticAsset('path/to/image/hello.png').path()
// starterCodePath is equal to 'path/to/image/hello.png'
```

のサブクラスは StaticAssetですSubstitutionAsset。サブクラス関数はまったく同じですが、代わりに ファイルに対してマスターチ置換を実行できます。 copy-and-replace スタイル生成の実行に役立ちます。

静的アセット置換は、生成されたソースリポジトリにシードされる静的ファイルをレンダリングするために、マスターシュテンプレートエンジンを使用します。Mustache テンプレートルールはレンダリング中に適用されます。つまり、すべての値はデフォルトで HTML エンコードされます。エスケープされていない HTML をレンダリングするには、トリプルマスターチ構文 を使用します{{{name}}}。詳細については、「 マスターシュテンプレートルール」を参照してください。

> **Note**
>
> テキストで解釈できないファイルの代わりに を実行すると、エラーが発生する可能性があります。

```
const starterCodeText = new SubstitionAsset('path/to/file/file.txt').subsitite({
  'my_variable': 'subbed value1',
  'another_variable': 'subbed value2'
})
```

複数のファイルをターゲットにする

静的アセットは、 の静的関数StaticAssetとそのサブクラスを介した glob ターゲットをサポートしています。これによりfindAll(...)、パス、コンテンツなどがプリロードされた静的アセットのリストが返されます。リストをFile構成と連鎖させて、内容を static-assets ディレクトリにコピーして貼り付けることができます。

```
new File(repository, `path/to/my/file/in/repo/file.img`, new Buffer(...));

new File(repository, `path/to/my/file/in/repo/new-img.img`, new StaticAsset('path/to/
image.png').content());
```

新しいリポジトリの作成とファイルの追加

リポジトリコンポーネントを使用して、生成されたプロジェクトに新しいリポジトリを作成できます。その後、作成したリポジトリにファイルまたはワークフローを追加できます。

```
import { SourceRepository } from '@amazon-codecatalyst/codecatalyst-source-
repositories';
...
const repository = new SourceRepository(this, { title: 'myRepo' });
```

次の例は、既存のリポジトリにファイルとワークフローを追加する方法を示しています。

```
import { SourceFile } from '@amazon-codecatalyst/codecatalyst-source-repositories';
import { Workflow } from '@amazon-codecatalyst/codecatalyst-workflows';
...
new SourceFile(repository, 'README.md', 'This is the content of my readme');
new Workflow(this, repository, {/**...workflowDefinition...**/});
```

2 つのコードを組み合わせるmyRepoと、 という名前のリポジトリが 1 つ生成され、ソースファイルREADME.mdと CodeCatalystワークフローがルートに表示されます。

設計図へのワークフローコンポーネントの追加

ワークフローは、トリガーに基づいてアクションを実行するために Amazon CodeCatalyst プロジェクトによって使用されます。ワークフローコンポーネントを使用して、ワークフロー YAML ファイルを構築してまとめることができます。詳細については、「ワークフロー YAML 定義」を参照してください。

Amazon CodeCatalyst ブループリントワークフローコンポーネントをインポートするには

blueprint.ts ファイルで、以下を追加します。

```
import { WorkflowBuilder, Workflow } from '@amazon-codecatalyst/codecatalyst-workflows'
```

トピック

- ワークフローコンポーネントの例
- 環境への接続

ワークフローコンポーネントの例

WorkflowBuilder コンポーネント

クラスを使用してワークフロー定義を構築できます。定義は、リポジトリでレンダリングするためのワークフローコンポーネントに渡すことができます。

```
import { WorkflowBuilder } from '@amazon-codecatalyst/codecatalyst-workflows'

const workflowBuilder = new WorkflowBuilder({} as Blueprint, {
  Name: 'my_workflow',
});

// trigger the workflow on pushes to branch 'main'
workflowBuilder.addBranchTrigger(['main']);

// add a build action
workflowBuilder.addBuildAction({
  // give the action a name
  actionName: 'build_and_do_some_other_stuff',

  // the action pulls from source code
  input: {
    Sources: ['WorkflowSource'],
  },

  // the output attempts to autodiscover test reports, but not in the node modules
  output: {
    AutoDiscoverReports: {
      Enabled: true,
      ReportNamePrefix: AutoDiscovered,
      IncludePaths: ['**/*'],
      ExcludePaths: ['*/node_modules/**/*'],
    },
  },
  // execute some arbitrary steps
  steps: [
    'npm install',
    'npm run myscript',
    'echo hello-world',
  ],
  // add an account connection to the workflow
  environment: convertToWorkflowEnvironment(myEnv),
```

```
});
```

Workflow Projen コンポーネント

次の例は、Projen コンポーネントを使用してワークフロー YAML をリポジトリに書き込む方法を示しています。

```
import { Workflow } from '@amazon-codecatalyst/codecatalyst-workflows'

...

const repo = new SourceRepository
const blueprint = this;
const workflowDef = workflowBuilder.getDefinition()

// creates a workflow.yaml at .aws/workflows/${workflowDef.name}.yaml
new Workflow(blueprint, repo, workflowDef);

// can also pass in any object and have it rendered as a yaml. This is unsafe and may
 not produce a valid workflow
new Workflow(blueprint, repo, {... some object ...});
```

環境への接続

多くのワークフローは、AWS アカウント接続で実行する必要があります。ワークフローは、アクションがアカウントとロール名の仕様を持つ環境に接続できるようにすることで、これを処理します。

```
import { convertToWorkflowEnvironment } from '@amazon-codecatalyst/codecatalyst-
workflows'


const myEnv = new Environment(...);

// can be passed into a workflow constructor
const workflowEnvironment = convertToWorkflowEnvironment(myEnv);


// add a build action
workflowBuilder.addBuildAction({
  ...
  // add an account connection to the workflow
```

```
  environment: convertToWorkflowEnvironment(myEnv),
});
```

開発環境コンポーネントを設計図に追加する

マネージド開発環境 (MDE) は、 で MDE Workspace を作成して起動するために使用されます CodeCatalyst。コンポーネントは devfile.yaml ファイルを生成します。詳細については、「開発ファイルの概要」および「」を参照してください開発環境のリポジトリ devfile の編集。

```
new Workspace(this, repository, SampleWorkspaces.default);
```

Amazon CodeCatalyst ブループリントワークスペースコンポーネントをインポートするには

blueprint.ts ファイルで、以下を追加します。

```
import {...} from '@amazon-codecatalyst/codecatalyst-workspaces'
```

ブループリントへの問題コンポーネントの追加

では CodeCatalyst、機能、タスク、バグ、およびプロジェクトに関連するその他の作業をモニタリングできます。各作業は、問題と呼ばれる個別のレコードに保持されます。各問題には、説明、担当者、ステータス、およびその他のプロパティがあり、検索、グループ化、フィルタリングできます。デフォルトのビューを使用して問題を表示することも、カスタムフィルタリング、ソート、グループ化を使用して独自のビューを作成することもできます。問題に関連する概念の詳細については、問題の概念「」および「」を参照してくださいの問題のクォータ CodeCatalyst。

問題コンポーネントは、問題の JSON 表現を生成します。コンポーネントは ID フィールドを受け取り、問題定義を入力として受け取ります。

Amazon CodeCatalyst ブループリントの問題コンポーネントをインポートするには

blueprint.ts ファイルで、以下を追加します。

```
import {...} from '@amazon-codecatalyst/blueprint-component.issues'
```

トピック

- 問題コンポーネントの例

問題コンポーネントの例

問題の作成

```
import { Issue } from '@amazon-codecatalyst/blueprint-component.issues';
...
new Issue(this, 'myFirstIssue', {
  title: 'myFirstIssue',
  content: 'This is an example issue.',
});
```

優先度の高い問題の作成

```
import { Workflow } from '@amazon-codecatalyst/codecatalyst-workflows'
...
const repo = new SourceRepository
const blueprint = this;
const workflowDef = workflowBuilder.getDefinition()

// Creates a workflow.yaml at .aws/workflows/${workflowDef.name}.yaml
new Workflow(blueprint, repo, workflowDef);

// Can also pass in any object and have it rendered as a yaml. This is unsafe and may
 not produce a valid workflow
new Workflow(blueprint, repo, {... some object ...});
```

ラベルで優先度の低い問題を作成する

```
import { Issue } from '@amazon-codecatalyst/blueprint-component.issues';
...
new Issue(this, 'myThirdIssue', {
  title: 'myThirdIssue',
  content: 'This is an example of a low priority issue with a label.',
  priority: 'LOW',
  labels: ['exampleLabel'],
});
```

設計図ツールと CLI の使用

ブループリント CLI には、カスタムブループリントを管理および操作するためのツールが用意されています。

トピック

- 設計図ツールの使用
- イメージアップロードツール

設計図ツールの使用

設計図ツールを使用するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 開発環境を再開します。詳細については、「開発環境の再開」を参照してください。

   開発環境がない場合は、まず開発環境を作成する必要があります。詳細については、「開発環境の作成」を参照してください。
3. 動作中のターミナルで、次のコマンドを実行して設計図 CLI をインストールします。

   ```
   npm install -g @amazon-codecatalyst/blueprint-util.cli
   ```
4. blueprint.ts ファイルに、使用するツールを次の形式でインポートします。

   ```
   import { <tooling-function-name> } from '@amazon-codecatalyst/blueprint-util.cli/
   lib/<tooling-folder-name>/<tooling-file-name>;
   ```

   > **Tip**
   >
   > に移動CodeCatalyst blueprints GitHub repositoryして、使用するツールの名前を見つけることができます。

   イメージアップロードツールを使用する場合は、スクリプトに以下を追加します。

   ```
   import { uploadImagePublicly } from '@amazon-codecatalyst/blueprint-util.cli/lib/
   image-upload-tool/upload-image-to-aws';
   ```

   例

   - 公開関数を使用する場合は、スクリプトに以下を追加します。

```
import { publish } from '@amazon-codecatalyst/blueprint-util.cli/lib/publish/
publish';
```

- イメージアップロードツールを使用する場合は、スクリプトに以下を追加します。

```
import { uploadImagePublicly } from '@amazon-codecatalyst/blueprint-util.cli/lib/
image-upload-tool/upload-image-to-aws';
```

5. 関数を呼び出します。

   例:

   - 公開関数を使用する場合は、スクリプトに以下を追加します。

```
await publish(logger, config.publishEndpoint, {<your publishing options>});
```

   - イメージアップロードツールを使用する場合は、スクリプトに以下を追加します。

```
const {imageUrl, imageName} = await uploadImagePublicly(logger, 'path/to/
image'));
```

イメージアップロードツール

イメージアップロードツールを使用すると、独自のイメージを AWS アカウントの S3 バケットにアップロードし、そのイメージを の背後にパブリックに配布できます CloudFront。このツールは、ローカルストレージ (およびオプションのバケット名) 内のイメージパスを入力として受け取り、公開されているイメージに URL を返します。詳細については、「Amazon CloudFrontとは」および「Amazon S3 とは」を参照してください。

イメージアップロードツールを使用するには

1. ブループリント SDK とサンプルブループリントへのアクセスを提供するオープンソースブループリント GitHub リポジトリのクローンを作成します。動作中のターミナルで、次のコマンドを実行します。

```
git clone https://github.com/aws/codecatalyst-blueprints.git
```

2. 次のコマンドを実行して、ブループリント GitHub リポジトリに移動します。

```
cd codecatalyst-blueprints
```

3. 次のコマンドを実行して、依存関係をインストールします。

```
yarn && yarn build
```

4. 次のコマンドを実行して、最新のブループリント CLI バージョンがインストールされていることを確認します。

```
yarn upgrade @amazon-codecatalyst/blueprint-util.cli
```

5. イメージをアップロードする S3 バケットを使用して AWS アカウントにログインします。詳細については、「AWS CLI の設定」および「AWS コマンドラインインターフェイス からサインインする」を参照してください。
6. CodeCatalyst リポジトリのルートから次のコマンドを実行して、設計図 CLI で ディレクトリに移動します。

```
cd packages/utils/blueprint-cli
```

7. 次のコマンドを実行して、S3 バケットにイメージをアップロードします。

```
yarn blueprint upload-image-public <./path/to/your/image>
      <optional:optional-bucket-name>
```

イメージへの URL が生成されます。URL は、ディストリビューションの CloudFront デプロイに時間がかかるため、すぐには利用できません。ディストリビューションのステータスをチェックして、最新のデプロイステータスを取得します。詳細については、「 ディストリビューションの使用」を参照してください。

スナップショットテストによるインターフェイスの変更の評価

ブループリントの複数の設定で生成されたスナップショットテストがサポートされています。

ブループリントは、ブループリント作成者としてユーザーが提供する設定でのスナップショットテストをサポートします。設定は、設計図のルートにある defaults.json ファイルの上にマージされる部分的なオーバーライドです。スナップショットテストを有効にして設定すると、ビルドおよびテストプロセスによって特定の設定が合成され、合成された出力が参照スナップショットから変更されて

いないことを確認します。スナップショットテストコードを表示するには、CodeCatalyst「設計図 GitHub リポジトリ」を参照してください。

スナップショットテストを有効にするには

1. .projenrc.ts ファイルで、スナップショットを作成するファイル ProjenBlueprint を使用して入力オブジェクトを に更新します。例:

```
{
  ....
  blueprintSnapshotConfiguration: {
    snapshotGlobs: ['**', '!environments/**', '!aws-account-to-environment/**'],
  },
}
```

2. 設計図を再合成して、設計図プロジェクトに TypeScript ファイルを作成します。ソースファイルは Projen によって管理および再生成されるため、編集しないでください。以下のコマンドを使用します。

```
yarn projen
```

3. src/snapshot-configurations ディレクトリに移動して、空のオブジェクトを持つdefault-config.jsonファイルを表示します。ファイルを更新または独自のテスト設定で置き換えます。その後、各テスト設定はプロジェクトの defaults.json ファイルとマージされ、合成され、テスト時にスナップショットと比較されます。次のコマンドを使用してテストします。

```
yarn test
```

テストコマンドを初めて使用すると、 というメッセージが表示されますSnapshot Summary › NN snapshots written from 1 test suite。以降のテスト実行では、合成された出力がスナップショットから変更されていないことを確認し、 というメッセージが表示されますSnapshots: NN passed, NN total。

ブループリントを意図的に変更して別の出力を生成する場合は、次のコマンドを実行して参照スナップショットを更新します。

```
yarn test:update
```

スナップショットでは、合成された出力が各実行間で一定であることが想定されます。ブループリントで異なるファイルが生成された場合は、それらのファイルをスナップショットテストから除外する必要があります。ProjenBluerpint 入力blueprintSnapshotConfigurationオブジェクトの オブジェクトを更新して、 snapshotGlobsプロパティを追加します。snapshotGlobs プロパティは、スナップショット作成に含めるファイルまたは除外するファイルを決定する glob の配列です。

> **Note**
>
> グロブのデフォルトリストがあります。独自のリストを指定する場合は、デフォルトエントリを明示的に取り戻す必要がある場合があります。

スペースへのカスタムブループリントの発行

スペースのブループリントカタログにカスタムブループリントを作成する前に、スペースに公開する必要があります。公開前に CodeCatalyst コンソールでブループリントを表示することもできます。ブループリントのプレビューバージョンまたは通常バージョンを公開できます。

> **Important**
>
> 外部ソースからのブループリントパッケージを使用する場合は、それらのパッケージに伴うリスクを考慮してください。スペースに追加するカスタムブループリントとそれらが生成するコードは、お客様の責任となります。

トピック

- カスタムブループリントのプレビューバージョンの表示と公開
- カスタムブループリントの通常バージョンの表示と公開
- 指定されたスペースとプロジェクトでのカスタムブループリントの公開と適用

カスタムブループリントのプレビューバージョンの表示と公開

スペースのブループリントカタログに追加する場合は、カスタムブループリントのプレビューバージョンをスペースに公開できます。これにより、非プレビューバージョンをカタログに追加する前に、ブループリントをユーザーとして表示できます。プレビューバージョンでは、実際のバージョンを使用せずに公開できます。例えば、ある0.0.1バージョンで作業する場合、プレビューバー

ジョンを公開して追加できるため、2 番目のバージョンの新しい更新を公開して として追加できます`0.0.2`。

変更を加えたら、 `package.json` ファイルを実行してカスタムブループリントのパッケージを再構築し、変更をプレビューします。

カスタムブループリントのプレビューバージョンを表示および公開するには

1. 開発環境を再開します。詳細については、「開発環境の再開」を参照してください。
2. 開発環境で作業ターミナルを開きます。
3. (オプション) 作業ターミナルで、プロジェクトに必要な依存関係をまだインストールしていない場合は、インストールします。以下のコマンドを使用します。

   ```
   yarn
   ```

4. (オプション) `.projenrc.ts` ファイルに変更を加えた場合は、設計図を構築してプレビューする前に、プロジェクトの設定を再生成します。以下のコマンドを使用します。

   ```
   yarn projen
   ```

5. 次のコマンドを使用して、カスタムブループリントを再構築してプレビューします。次のコマンドを使用します。

   ```
   yarn blueprint:preview
   ```

   提供されている`See this blueprint at:`リンクに移動して、カスタムブループリントをプレビューします。設定に基づいて、テキストを含む UI が想定どおりに表示されることを確認します。カスタムブループリントを変更する場合は、`blueprint.ts`ファイルを編集し、ブループリントを再合成してから、プレビューバージョンを再度公開できます。詳細については、「再合成」を参照してください。
6. (オプション) カスタムブループリントのプレビューバージョンを公開し、スペースのブループリントカタログに追加できます。`Enable version` *`[preview version number]`* `at:` リンクに移動して、プレビューバージョンをスペースに公開します。

でプロジェクトを作成することなく、プロジェクトの作成をエミュレートできます CodeCatalyst。プロジェクトを合成するには、次のコマンドを使用します。

```
yarn blueprint:synth
```

設計図が synth/synth.*[options-name]*/proposed-bundle/フォルダに生成されます。詳細については、「合成」を参照してください。

カスタムブループリントを更新する場合は、代わりに次のコマンドを使用してプロジェクトを再合成します。

```
yarn blueprint:resynth
```

設計図が synth/synth.*[options-name]*/proposed-bundle/フォルダに生成されます。詳細については、「再合成」を参照してください。

プレビューバージョンを公開したら、スペースメンバーがそれを使用して新しいプロジェクトを作成したり、既存のプロジェクトに適用したりできるように、ブループリントを追加できます。詳細については、「スペースカタログへのカスタムブループリントの追加」を参照してください。

カスタムブループリントの通常バージョンの表示と公開

カスタムブループリントの開発とプレビューが完了したら、スペースのブループリントカタログに追加する新しいバージョンを表示して公開できます。プロジェクトの作成時に生成されるリリースワークフローは、プッシュされる変更を自動的に発行します。ブループリントの作成時にワークフロー生成をオフにした場合、ブループリントは自動的にスペースのブループリントカタログに追加されません。yarn コマンドを実行した後も、カスタムブループリントをスペースに公開できます。

カスタムブループリントを表示して公開するには

1. 開発環境を再開します。詳細については、「開発環境の再開」を参照してください。
2. 開発環境で作業ターミナルを開きます。
3. • ブループリントの作成時にリリースワークフローの生成をオプトアウトした場合は、次のコマンドを使用します。

   ```
   yarn blueprint:release
   ```

   提供されているSee this blueprint at:リンクに移動して、カスタムブループリントを表示できます。

   カスタムブループリントの更新バージョンを公開し、スペースのブループリントカタログに追加できます。Enable version *[release version number]* at: リンクに移動して、最新バージョンをスペースに公開します。

- ブループリントの作成時にリリースワークフローをオプトインした場合、変更がプッシュされると、最新のブループリントバージョンが自動的に発行されます。次のコマンドを使用します。

```
git add .
```

```
git commit -m "commit message"
```

```
git push
```

通常のバージョンを公開したら、スペースメンバーがブループリントを使用して新しいプロジェクトを作成したり、既存のプロジェクトに適用したりできるように、ブループリントを追加できます。詳細については、「スペースカタログへのカスタムブループリントの追加」を参照してください。

指定されたスペースとプロジェクトでのカスタムブループリントの公開と適用

デフォルトでは、 コマンドblueprint:previewと blueprint:release コマンドは、ブループリントを作成した CodeCatalyst スペースに発行されます。複数のエンタープライズスペースがある場合は、それらのスペースで同じブループリントをプレビューして公開することもできます。別のスペースの既存のプロジェクトにブループリントを適用することもできます。

指定したスペースでカスタムブループリントを公開または適用するには

1. 開発環境を再開します。詳細については、「開発環境の再開」を参照してください。
2. 開発環境で作業ターミナルを開きます。
3. (オプション) プロジェクトに必要な依存関係をまだインストールしていない場合は、インストールします。以下のコマンドを使用します。

```
yarn
```

4. --space タグを使用して、指定したスペースにプレビューまたは通常バージョンを公開します。例:

- ```
  yarn blueprint:preview --space my-awesome-space # publishes under a "preview"
   version tag to 'my-awesome-space'
  ```

  出力例:
  ```

```
Enable version 0.0.1-preview.0 at: https://codecatalyst.aws/spaces/my-awesome-
space/blueprints
Blueprint applied to [NEW]: https://codecatalyst.aws/spaces/my-awesome-space/
blueprints/%40amazon-codecatalyst%2Fmyspace.my-blueprint/publishers/1524817d-
a69b-4abe-89a0-0e4a9a6c53b2/versions/0.0.1-preview.0/projects/create
```

- ```
  yarn blueprint:release --space my-awesome-space # publishes normal version to
   'my-awesome-space'
  ```

  出力例:

  ```
  Enable version 0.0.1 at: https://codecatalyst.aws/spaces/my-awesome-space/
  blueprints
  Blueprint applied to [NEW]: https://codecatalyst.aws/spaces/my-awesome-space/
  blueprints/%40amazon-codecatalyst%2Fmyspace.my-blueprint/publishers/1524817d-
  a69b-4abe-89a0-0e4a9a6c53b2/versions/0.0.1/projects/create
  ```

を使用して、カスタムブループリントのプレビューバージョンを指定されたスペース内の既存のプロジェクト`--project`に適用します。例:

```
yarn blueprint:preview --space my-awesome-space --project my-project # previews
 blueprint application to an existing project
```

出力例:

```
Enable version 0.0.1-preview.1 at: https://codecatalyst.aws/spaces/my-awesome-
space/blueprints
Blueprint applied to [my-project]: https://codecatalyst.aws/spaces/my-awesome-
space/projects/my-project/blueprints/%40amazon-codecatalyst%2FmySpace.my-blueprint/
publishers/1524817d-a69b-4abe-89a0-0e4a9a6c53b2/versions/0.0.1-preview.1/add
```

カスタムブループリントの詳細、バージョン、プロジェクトの表示

ブループリントの詳細、バージョン、ブループリントを使用して作成または適用されたプロジェクトなど、スペースで公開されているカスタムブループリントを表示できます。

トピック
```

- スペースのカスタムブループリントを表示する
- カスタムブループリントで作成されたプロジェクトの表示、またはカスタムブループリントの適用

スペースのカスタムブループリントを表示する

スペースのカスタムブループリントを表示するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. CodeCatalyst コンソールで、カスタムブループリントを表示したいスペースに移動します。
3. スペースダッシュボードで [設定] タブを選択し、[ブループリント] を選択してスペースブループリントを表示します。テーブルには以下の詳細が表示されます。
   - 名前-カスタムブループリントの名前。
   - カタログステータス-カスタムブループリントがスペースのブループリントカタログに公開されているかどうか。
   - 最新バージョン-カスタムブループリントの最新バージョン。
   - 最終更新日-スペースブループリントが最後に更新された日付。

カスタムブループリントで作成されたプロジェクトの表示、またはカスタムブループリントの適用

カスタムブループリントを使用して作成されたプロジェクト、またはカスタムブループリントを適用したプロジェクトを表示するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. CodeCatalyst コンソールで、カスタムブループリントを表示したいスペースに移動します。
3. スペースダッシュボードで [設定] タブを選択し、[ブループリント] を選択します。
4. スペースブループリントテーブルからカスタムブループリントの名前を選択すると、ブループリントを使用しているプロジェクトとブループリントテーブルを使用していないプロジェクトが表示されます。

スペースカタログへのカスタムブループリントの追加

カスタムブループリントをスペースに公開すると、スペースのブループリントカタログに追加できます。 CodeCatalyst スペースのブループリントカタログにカスタムブループリントを追加すると、そ

のブループリントは、プロジェクトの作成時または既存のプロジェクトへの適用時に使用できるすべてのスペースメンバーが使用できます。スペースのブループリントカタログにカスタムブループリントを追加する前に、ブループリントの公開アクセス許可を有効にする必要があります。ワークフローリリース生成をオプトインした場合、パブリッシュ許可はデフォルトで有効になります。詳細については、「カスタムブループリントの発行許可の設定」および「スペースへのカスタムブループリントの発行」を参照してください。

スペースのブループリントカタログにブループリントを追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. ブループリントは、ソースリポジトリのデフォルトブランチからのみ追加できます。機能ブランチでブループリントを作成した場合は、機能ブランチをデフォルトのブランチへの変更とマージします。プルリクエストを作成して、変更をデフォルトのブランチにマージします。詳細については、「Amazon でのプルリクエストによるコードの確認 CodeCatalyst」を参照してください。
3. CodeCatalyst コンソールで、カスタムブループリントを使用してスペースダッシュボードに移動します。
4. スペースダッシュボードで、設定タブを選択し、ブループリント を選択します。
5. 追加するブループリント名を選択し、カタログに追加 を選択します。複数のバージョンがある場合は、カタログバージョンドロップダウンメニューからバージョンを選択します。
6. [保存] を選択します。

スペースカタログからのカスタムブループリントの削除

カスタムブループリントは、新しいプロジェクトの作成に使用したり、既存のプロジェクトに適用したりしたくない場合は、スペースのブループリントカタログから削除できます。

> **Note**
>
> スペースカタログからカスタムブループリントを削除しても、ブループリントから作成されたプロジェクトや、ブループリントを適用したプロジェクトには影響しません。設計図のリソースはプロジェクトから削除されません。

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。

2. CodeCatalyst コンソールで、カスタムブループリントを使用してスペースダッシュボードに移動します。
3. スペースダッシュボードで、設定タブを選択し、ブループリント を選択します。
4. 削除するブループリント名を選択し、カタログ からブループリントを削除を選択します。

カスタムブループリントの発行許可の設定

デフォルトでは、カスタムブループリントの アクセス許可は、プロジェクトの作成中にワークフローリリースが生成された場合に有効になります。パブリッシュ許可を有効にすると、ブループリントをスペースにパブリッシュできます。アクセス許可を無効にすると、ブループリントを公開できなくなります。アクセス許可が無効になっている場合、ブループリントの作成中に生成されるリリースワークフローは実行されません。ブループリントに対する新しい変更は、ブループリントのアクセス許可が有効になっていない限り公開できません。

> **Important**
>
> ブループリントプロジェクトの公開許可を有効または無効にするには、スペース管理者ロールが必要です。

ブループリントプロジェクトの公開アクセス許可を設定するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. CodeCatalyst コンソールで、カスタムブループリントの公開アクセス許可を管理するスペースに移動します。
3. スペースダッシュボードで、設定タブを選択し、ブループリント を選択します。
4. プロジェクト発行許可タブを選択すると、すべてのスペースのブループリントに対する発行許可が表示されます。
5. 管理するブループリントを選択し、有効化または無効化を選択して発行許可を変更します。アクセス許可を有効にする場合は、アクセス許可の変更の詳細を確認し、ブループリントの公開を有効にするを選択して変更を確認します。

カスタムブループリントのカタログバージョンの変更

ブループリントの作成者は、スペースのブループリントカタログに公開するバージョンを管理できます。設計図のカタログバージョンを変更しても、別の設計図バージョンを使用しているプロジェクトには影響しません。

カスタムブループリントバージョンを管理するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. CodeCatalyst コンソールで、カスタムブループリントのバージョンを変更するスペースに移動します。
3. スペースダッシュボードで、設定タブを選択し、ブループリント を選択します。
4. スペースブループリントテーブルで、管理するカスタムブループリントのラジオボタンを選択します。
5. カタログバージョンを作成 を選択し、カタログバージョンドロップダウンメニューからバージョンを選択します。
6. [保存] を選択します。

公開済みのカスタムブループリントまたはバージョンの削除

カスタムブループリントのバージョンまたはブループリント自体を Amazon CodeCatalyst スペースから削除すると、ブループリントプロジェクトまたはブループリントバージョンのリソースへのすべてのアクセス権が削除されます。ブループリントバージョンまたはブループリントを削除すると、プロジェクトメンバーはプロジェクトリソースにアクセスできなくなり、サードパーティのソースリポジトリによって要求されたワークフローはすべて停止されます。

> **Note**
>
> ブループリントを削除しても、ブループリントが適用されているプロジェクトには影響しません。ブループリントのリソースはプロジェクトから削除されません。

ブループリントバージョンがスペースのブループリントカタログに公開されている場合は、公開済みバージョンを削除する前に、カタログの新しいバージョンを選択してください。

カスタムブループリントのカタログバージョンを削除するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. CodeCatalyst コンソールで、カスタムブループリントのカタログバージョンを削除したいスペースに移動します。
3. スペースダッシュボードで [設定] タブを選択し、次に [ブループリント] を選択します。
4. 削除するブループリントの名前とカタログバージョンを選択します。
5. 削除するカタログバージョンのラジオボタンを選択し、[バージョンを削除] を選択します。
6. 詳細を確認し、[新しいブループリントカタログバージョンの選択] ドロップダウンメニューから別のブループリントバージョンを選択します。
7. `delete`と入力して、ブループリントカタログバージョンの削除を確認します。
8. [削除] をクリックします。

ブループリントバージョンがスペースのブループリントカタログにない場合は、新しいバージョンを選択せずにそのバージョンを削除できます。

カスタムブループリントバージョンを削除するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. CodeCatalyst コンソールで、カスタムブループリントバージョンを削除したいスペースに移動します。
3. スペースダッシュボードで [設定] タブを選択し、次に [ブループリント] を選択します。
4. 削除するバージョンを含むブループリントの名前を選択します。
5. 削除するバージョンのラジオボタンを選択し、[Delete version] を選択します。
6. `delete`と入力してブループリントバージョンの削除を確認します。
7. [削除] をクリックします。

スペースのブループリントカタログからブループリントを削除すると、ブループリントのすべてのバージョンが削除されます。ブループリントを使用しているスペースのプロジェクトは、削除の影響を受けません。

カスタムブループリントバージョンを削除するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. CodeCatalyst コンソールで、カスタムブループリントを削除したいスペースに移動します。
3. スペースダッシュボードで [設定] タブを選択し、[ブループリント] を選択します。

4. スペースブループリントテーブルで、削除するカスタムブループリントのラジオボタンを選択し、次に [ブループリントを削除] を選択します。
5. deleteと入力してカスタムブループリントの削除を確定します。
6. [Delete] (削除) をクリックします。

依存関係、不一致、ツールの処理

トピック

- 依存関係の追加
- 依存関係タイプの不一致の処理
- yarn と npm の使用
- ツールとコンポーネントのアップグレード

依存関係の追加

設計図の作成者として、 などのパッケージを設計図に追加する必要がある場合があります`@amazon-codecatalyst/blueprint-component.environments`。そのパッケージで`projen.ts`ファイルを更新し、Projen を使用してプロジェクトの設定を再生成する必要があります。Projen は、各ブループリントコードベースのプロジェクトモデルとして機能し、モデルが設定ファイルをレンダリングする方法を変更することで、下位互換性のあるツールの更新をプッシュする機能を提供します。package.json ファイルは Projen モデルによって部分的に所有されているファイルです。Projen は package.json ファイルに含まれる依存関係バージョンを承認しますが、他のオプションはモデルから取得する必要があります。

依存関係を追加して**`projenrc.ts`**ファイルを更新するには

1. projen.ts ファイルで、deps セクションに移動します。
2. 設計図で使用する依存関係を追加します。
3. 次のコマンドを使用して、プロジェクトの設定を再生成します。

```
yarn projen && yarn
```

依存関係タイプの不一致の処理

Yarn 更新後、リポジトリパラメータに関して次のエラーが表示されることがあります。

```
Type 'SourceRepository' is missing the following properties from type
 'SourceRepository': synthesisSteps, addSynthesisStep
```

エラーは、あるコンポーネントが別のコンポーネントの新しいバージョンに依存しているが、依存するコンポーネントが古いバージョンに固定されている場合に発生する依存関係の不一致が原因です。エラーは、すべてのコンポーネントが同じバージョンに依存して、バージョンが同期されるようにすることで修正できます。バージョンの処理方法が明確でない限り、設計図で提供されるパッケージを同じ最新バージョン (`0.0.x`) に維持することをお勧めします。次の例は、すべての依存関係が同じバージョンに依存するように `package.json` ファイルを設定する方法を示しています。

```
...
"@caws-blueprint-component/caws-environments": "^0.1.12345",
"@caws-blueprint-component/caws-source-repositories": "^0.1.12345",
"@caws-blueprint-component/caws-workflows": "^0.1.12345",
"@caws-blueprint-component/caws-workspaces": "^0.1.12345",
"@caws-blueprint-util/blueprint-utils": "^0.1.12345",
...
"@caws-blueprint/blueprints.blueprint": "*",
```

すべての依存関係のバージョンを設定したら、次のコマンドを使用します。

```
yarn install
```

yarn と npm の使用

設計図では、ツールに Yarn を使用しています。npm と Yarn を使用すると、依存関係ツリーの解決方法がそれぞれ異なるため、ツールの問題が発生します。このような問題を回避するには、Yarn のみを使用することをお勧めします。

npm を使用して依存関係を誤ってインストールした場合は、生成された`package-lock.json`ファイルを削除し、必要な依存関係で`.projenrc.ts`ファイルが更新されていることを確認します。Projen を使用してプロジェクトの設定を再生成します。

モデルから再生成するには、以下を使用します。

```
yarn projen
```

.projenrc.ts ファイルが必要な依存関係で更新されていることを確認したら、次のコマンドを使用します。

```
yarn
```

ツールとコンポーネントのアップグレード

場合によっては、ツールやコンポーネントをアップグレードして、利用可能な新機能を導入することもできます。バージョンの処理方法が明確でない限り、すべてのコンポーネントを同じバージョンに保つことをお勧めします。バージョンはコンポーネント間で同期されるため、すべてのコンポーネントで同じバージョンを使用することで、コンポーネント間の適切な依存関係が確保されます。

Yarn ワークスペースモノレポの使用

次のコマンドを使用して、カスタムブループリントのリポジトリのルートから utils とコンポーネントをアップグレードします。

```
yarn upgrade @amazon-codecatalyst/*
```

monorepo を使用していない場合は、次のコマンドを使用します。

```
yarn upgrade –pattern @amazon-codecatalyst/*
```

ツールとコンポーネントのアップグレードに使用できるその他のオプション：

- npm ビュー`@caws-blueprint-component/<some-component>`を使用して最新バージョンを取得します。
- package.json ファイルでバージョンを設定し、次のコマンドを使用して、 を最新バージョンに手動で増やします`yarn`。すべてのコンポーネントと utils のバージョンが同じである必要があります。

説明

ブループリントソフトウェア開発キット (SDK) は、提供できるオープンソースライブラリです。寄稿者として、寄稿ガイドライン、フィードバック、欠陥を考慮してください。詳細については、「ブループリント GitHub リポジトリ」を参照してください。

のブループリントのクォータ CodeCatalyst

次の表に、Amazon のブループリントのクォータと制限を示します CodeCatalyst。Amazon のクォータの詳細については CodeCatalyst、「」を参照してくださいのクォータ CodeCatalyst。

CodeCatalyst プロジェクトごとに適用されるブループリントの最大数	100

でソースリポジトリを使用してコードを保存し、共同作業する CodeCatalyst

CodeCatalyst ソースリポジトリは、Amazon でホストされている Git リポジトリです CodeCatalyst。でソースリポジトリを使用して、プロジェクトのアセット CodeCatalyst を安全に保存、バージョン管理できます。

CodeCatalyst リポジトリ内のアセットには以下が含まれます。

- ドキュメント
- ソースコード
- バイナリファイル

CodeCatalyst また、 はプロジェクトのソースリポジトリを使用して、ワークフロー設定ファイルなどのプロジェクトの設定情報を保存します。

CodeCatalyst プロジェクトには複数のソースリポジトリを含めることができます。例えば、フロントエンドのソースコード、バックエンドのソースコード、ユーティリティ、ドキュメント用に個別のソースリポジトリを用意できます。

ソースリポジトリ、プルリクエスト、および の開発環境でコードを操作するためのワークフローの 1 つを以下に示します CodeCatalyst。

Mary Major は、ブループリント CodeCatalyst を使用して にウェブアプリケーションプロジェクトを作成し、サンプルコードを含むソースリポジトリを作成します。彼女は、友人の Li Juan、Saanvi Sarkar、Jorge Souza を招待して、彼女と一緒にプロジェクトに取り組みます。Li Juan はソースリポジトリのサンプルコードを調べ、コードにテストを追加するためにいくつかの簡単な変更を加えることにしました。Li は開発環境を作成し、IDE AWS Cloud9 として を選択し、新しいブランチである###### を指定します。開発環境が開きます。Li はコードをすばやく追加し、変更を加えてブランチをコミットして のソースリポジトリにプッシュします CodeCatalyst。次に、Li はプルリクエストを作成します。このプルリクエストの作成の一環として、Li はコードがレビューされていることを確認するためにレビューワーとして Jorge Souza と Saanvi Sarkar を追加します。

コードを確認すると、Jorge Souza は、作業 GitHub しているアプリケーションのプロトタイプを含む独自のプロジェクトリポジトリが にあることを覚えています。彼は Mary Major に、リポジトリを追加のソース GitHub リポジトリとしてプロジェクトにリンクできる 拡張機能をインストールして設定するよう依頼します。Mary は のリポジトリを確認し GitHub 、Jorge と協力して GitHub 拡張機能

を設定し、 GitHub リポジトリをプロジェクトの追加のソースリポジトリとしてリンクできるようにします。

CodeCatalyst ソースリポジトリは Git の標準機能をサポートし、既存の Git ベースのツールと連携します。Git クライアントまたは統合開発環境 (IDE) からソースリポジトリをクローンして操作するときに、アプリケーション固有のパスワードとして個人用アクセストークン (PATs) を作成して使用できます。 IDEs これらの PATsはユーザー ID に関連付けられています CodeCatalyst。詳細については、「個人用アクセストークンを使用してリポジトリアクセスをユーザーに付与する」を参照してください。

CodeCatalyst ソースリポジトリはプルリクエストをサポートします。これは、あるブランチから別のブランチにマージする前に、ユーザーや他のプロジェクトメンバーがコードの変更を確認してコメントするための簡単な方法です。 CodeCatalyst コンソールで変更を表示し、コード行にコメントできます。

CodeCatalyst ソースリポジトリのブランチにプッシュすると、変更を構築、テスト、デプロイできるワークフローで自動的に実行を開始できます。ソースリポジトリがプロジェクトテンプレートを使用してプロジェクトの一部として作成された場合、プロジェクトの一部として 1 つ以上のワークフローが設定されます。リポジトリのワークフローはいつでも追加できます。プロジェクト内のワークフローの YAML 設定ファイルは、それらのワークフローのソースアクションで設定されたソースリポジトリに保存されます。詳細については、「ワークフローの開始方法」を参照してください。

トピック

- ソースリポジトリの概念
- ソースリポジトリを操作するための のセットアップ
- CodeCatalyst ソースリポジトリと単一ページのアプリケーション設計図の開始方法
- のプロジェクトのリポジトリにソースコードを保存する CodeCatalyst
- ソースコードを整理して Amazon のブランチを操作する CodeCatalyst
- Amazon でのソースコードファイルの管理 CodeCatalyst
- Amazon でのプルリクエストによるコードの確認 CodeCatalyst
- Amazon でのコミットによるソースコードの変更について CodeCatalyst
- のソースリポジトリのクォータ CodeCatalyst

ソースリポジトリの概念

CodeCatalyst ソースリポジトリを使用する際に知っておくべき概念をいくつか紹介します。

トピック

プロジェクト

プロジェクトは、開発チームやタスクをサポートする CodeCatalyst での共同作業を表します。プロジェクトを作成したら、ユーザーとリソースの追加、更新、削除、プロジェクトダッシュボードのカスタマイズ、チームの作業の進行状況のモニタリングを行うことができます。1 つのスペースに複数のプロジェクトを含めることができます。

ソースリポジトリは、スペース内で作成またはリンクするプロジェクトに固有です。プロジェクト間でリポジトリを共有したり、リポジトリをスペース内の複数のプロジェクトにリンクしたりすることはできません。プロジェクトに Contributor またはプロジェクト管理者ロールを持つユーザーは、それらのロールに付与されたアクセス許可に従って、そのプロジェクトに関連付けられたソースリポジトリとやり取りできます。詳細については、「ユーザーロールによるアクセス許可の付与」を参照してください。

ソースリポジトリ

ソースリポジトリは、プロジェクトのコードとファイルを安全に保存する場所です。また、ファイルのバージョン履歴も保存されます。デフォルトでは、ソースリポジトリは CodeCatalyst プロジェクトの他のユーザーと共有されます。プロジェクトには複数のソースリポジトリを含めることができます。でプロジェクトのソースリポジトリを作成することも CodeCatalyst、インストールされている拡張機能でそのサービスがサポートされている場合は、別のサービスによってホストされている既存のソースリポジトリをリンクすることもできます。例えば、 GitHub リポジトリ拡張機能をインストールした後、GitHub リポジトリをプロジェクトにリンクできます。詳細については、「のプロ

ジェクトのリポジトリにソースコードを保存する CodeCatalyst」および「クイックスタート: 拡張機能のインストール、プロバイダーの接続、 でのリソースのリンク CodeCatalyst」を参照してください。

開発環境

開発環境は、 で使用 CodeCatalyst して、プロジェクトのソースリポジトリに保存されているコードをすばやく処理できるクラウドベースの開発環境です。開発環境に含まれるプロジェクトツールとアプリケーションライブラリは、プロジェクトのソースリポジトリ内の devfile によって定義されます。ソースリポジトリに devfile がない場合、デフォルトの devfile が自動的に適用されます。デフォルトの devfile には、最も頻繁に使用されるプログラミング言語とフレームワーク用のツールが含まれています。デフォルトでは、開発環境は 2 コアプロセッサ、4 GB の RAM、16 GiB の永続ストレージを持つように設定されています。

ソースリポジトリの既存のブランチを開発環境にクローンするか、開発環境の作成の一環として新しいブランチを作成するかを選択できます。

個人用アクセストークン (PATs)

個人アクセストークン (PAT) はパスワードに似ています。これは、 のすべてのスペースとプロジェクトで使用できるようにユーザー ID に関連付けられています CodeCatalyst。PATs、統合開発環境 (IDEsと Git ベースのソースリポジトリを含む CodeCatalyst リソースにアクセスします。PATs でユーザーを表 CodeCatalyst し、ユーザー設定で管理できます。ユーザーは複数の PAT を持つことができます。個人用アクセストークンは 1 回だけ表示されます。ベストプラクティスとして、ローカルコンピュータに安全に保存してください。デフォルトでは、PATs 1 年後に期限切れになります。

統合開発環境 (IDEsを使用する場合、PATs は Git パスワードと同等です。Git リポジトリと連携するように IDE を設定するときにパスワードを求められたら、PAT を指定します。IDE を Git ベースのリポジトリに接続する方法の詳細については、IDE のドキュメントを参照してください。

ブランチ

ブランチは、Git および のコミットへのポインタまたは参照です CodeCatalyst。ブランチを使用して作業を整理できます。例えば、ブランチを使用して、他のブランチのファイルに影響を与えずに、新しいバージョンまたは異なるバージョンのファイルで作業できます。ブランチを使用して、新機能の開発、プロジェクトの特定のバージョンの保存などを行うことができます。ソースリポジトリには、1 つのブランチまたは複数のブランチを含めることができます。テンプレートを使用してプロ

ジェクトを作成すると、プロジェクト用に作成されたソースリポジトリには、main というブランチにサンプルファイルが含まれます。メインブランチはリポジトリのデフォルトブランチです。

デフォルトのブランチ

のソースリポジトリ CodeCatalyst には、作成方法に関係なくデフォルトのブランチがあります。テンプレートを使用してプロジェクトを作成する場合、そのプロジェクト用に作成されたソースリポジトリには、サンプルコード、ワークフロー定義、その他のリソースに加えて README.md ファイルが含まれます。テンプレートを使用せずにソースリポジトリを作成すると、README.md ファイルが最初のコミットとして追加され、リポジトリの作成の一環としてデフォルトのブランチが作成されます。このデフォルトのブランチは main という名前です。このデフォルトブランチは、ユーザーがリポジトリをクローンするときに、ローカルリポジトリのベースブランチまたはデフォルトブランチとして使用されるブランチです。デフォルトブランチとして使用されるブランチを変更できます。詳細については、「リポジトリのデフォルトブランチの管理」を参照してください。

ソースリポジトリのデフォルトブランチは削除できません。検索結果には、デフォルトのブランチの結果のみが含まれます。

コミット

コミットとは、ファイルまたはファイルのセットに対する変更です。Amazon CodeCatalyst コンソールでは、コミットによって変更が保存され、ソースリポジトリにプッシュされます。コミットには、変更を行ったユーザーのアイデンティティ、変更日時、コミットタイトル、変更に関するメッセージなど、変更に関する情報が含まれます。詳細については、「Amazon でのコミットによるソースコードの変更について CodeCatalyst」を参照してください。

のソースリポジトリのコンテキストでは CodeCatalyst、コミットはコンテンツのスナップショットであり、リポジトリの内容に対する変更です。コミットに Git タグを追加して、特定のコミットを識別することもできます。

プルリクエスト

プルリクエストは、ユーザーや他のユーザーがソースリポジトリ内のブランチ間でコードの変更を確認、コメント、マージする主な方法です。プルリクエストを使用して、リリースされたソフトウェアのマイナーな変更や修正、主要な機能の追加、または新しいバージョンについて、コードの変更を共同で確認できます。プルリクエストでは、送信元ブランチと送信先ブランチの変更、またはそれらのブランチのリビジョンの違いを確認できます。個々のコード変更行にコメントを追加したり、プルリクエスト全体に対するコメントを追加したりできます。

> **Tip**
>
> プルリクエストの作成中に表示される違いは、ソースブランチのチップと宛先ブランチのチップの違いです。プルリクエストが作成されると、表示される違いは、選択したプルリクエストのリビジョンと、プルリクエストの作成時に送信先ブランチのヒントであったコミットの間になります。Git の違いとマージベースの詳細については、Git ドキュメントgit-merge-baseの「」を参照してください。

リビジョン

リビジョンは、プルリクエストの更新バージョンです。プルリクエストの送信元ブランチへのプッシュごとに、そのプッシュに含まれるコミットに加えられた変更を含むリビジョンが作成されます。プルリクエストのリビジョン間の違いは、送信元ブランチと送信先ブランチの違いに加えて表示できます。詳細については、「Amazon でのプルリクエストによるコードの確認 CodeCatalyst」を参照してください。

ワークフロー

ワークフローは、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) システムの一部としてコードを構築、テスト、デプロイする方法を説明する自動化された手順です。ワークフローは、ワークフローの実行中に実行する一連のステップまたはアクションを定義します。ワークフローは、ワークフローを開始するイベント、またはトリガー も定義します。ワークフローを設定するには、CodeCatalyst コンソールのビジュアルまたは YAML エディタ を使用してワークフロー定義ファイルを作成します。

> **Tip**
>
> プロジェクトでワークフローを使用する方法を簡単に確認するには、設計図 を使用してプロジェクトを作成します。各ブループリントは、レビュー、実行、および実験できる機能するワークフローをデプロイします。

ソースリポジトリは、プロジェクトのワークフロー、通知、問題、その他の設定情報の設定ファイルやその他の情報を保存することもできます。設定ファイルは、設定ファイルを必要とするリソースを作成するとき、またはワークフローのソースアクションとしてリポジトリを指定するときに、ソースリポジトリに作成および保存されます。設計図からプロジェクトを作成する場合、プロジェクトの一部として作成されたソースリポジトリに設定ファイルが既に保存されています。この設定情報は、リ

ポジトリのデフォルトブランチ.codecatalystの という名前のフォルダに保存されます。デフォルトブランチのブランチを作成するときは、そのブランチ内の他のすべてのファイルとフォルダに加えて、このフォルダとその設定のコピーを作成します。

ソースリポジトリを操作するための のセットアップ

CodeCatalyst ローカルマシンで Amazon のソースリポジトリを操作する場合、Git を単独で、またはサポートされている統合開発環境 (IDE) で使用して、コードを変更し、コードをプッシュおよびプルできます。ベストプラクティスとして、最新バージョンの Git やその他のソフトウェアを使用することをお勧めします。

> **Note**
>
> 開発環境を使用する場合は、Git をインストールする必要はありません。Git の最新バージョンが開発環境に含まれています。

のバージョン互換性情報 CodeCatalyst

コンポーネント	バージョン
Git	最新

Git をインストールする

IDE を使用せずに Git クライアントからソースリポジトリ内のファイル、コミット、ブランチ、その他の情報を操作するには、ローカルマシンに Git をインストールします。

Git をインストールするには、Git のダウンロードなどのウェブサイトをお勧めします。

個人用アクセストークンを作成する

ソースリポジトリをローカルマシンまたは任意の IDE にクローンするには、個人用アクセストークン (PAT) を作成する必要があります。

個人アクセストークン (PAT) を作成するには

1. 上部のメニューバーでプロファイルバッジを選択し、[My 設定] を選択します。

> **Tip**
>
> ユーザープロファイルは、プロジェクトまたはスペースのメンバーページに移動し、メンバーリストから自分の名前を選択することで検索することもできます。

2. PAT 名 に、PAT のわかりやすい名前を入力します。
3. 有効期限 で、デフォルトの日付のままにするか、カレンダーアイコンを選択してカスタム日付を選択します。有効期限は、デフォルトで現在の日付から 1 年です。
4. [作成] を選択します。

   このトークンは、ソースリポジトリの [クローンリポジトリC] を選択したときにも作成できます。
5. PAT シークレットを安全な場所に保存します。

> **Important**
>
> PAT シークレットは 1 回だけ表示されます。ウィンドウを閉じた後に取得することはできません。

CodeCatalyst ソースリポジトリと単一ページのアプリケーション設計図の開始方法

このチュートリアルのステップに従って、Amazon でソースリポジトリを操作する方法を学びます CodeCatalyst。

Amazon でソースリポジトリの使用を開始する最も簡単な方法は、テンプレートを使用してプロジェクトを作成する CodeCatalyst ことです。テンプレートを使用してプロジェクトを作成すると、サンプルコードを含むソースリポジトリを含むリソースが自動的に作成されます。このリポジトリとコード例を使用して、次の方法について説明します。

- プロジェクトのソースリポジトリを表示し、その内容を参照する
- コードを操作できる新しいブランチを使用して開発環境を作成する
- ファイルを変更し、変更をコミットしてプッシュする
- プルリクエストを作成し、他のプロジェクトメンバーとコードの変更を確認します。

- プルリクエストのソースブランチの変更を自動的に構築してテストするプロジェクトのワークフローを参照してください。
- 送信元ブランチから送信先ブランチに変更をマージし、プルリクエストを閉じます。
- 自動的に構築およびデプロイされたマージされた変更を確認する

このチュートリアルを最大限に活用するには、他のユーザーをプロジェクトに招待して、プルリクエストで協力できるようにします。また、問題の作成やプルリクエストへの関連付け CodeCatalyst、関連するワークフローの実行時の通知の設定やアラートの取得など、 の追加機能を調べることもできます。の詳細については、 CodeCatalyst「」を参照してください入門チュートリアル。

設計図を使用したプロジェクトの作成

プロジェクトの作成は、連携するための最初のステップです。ブループリントを使用してプロジェクトを作成できます。これにより、サンプルコードを含むソースリポジトリと、変更時にコードを自動的に構築してデプロイするワークフローも作成されます。このチュートリアルでは、単一ページのアプリケーションブループリントで作成されたプロジェクトについて説明しますが、ソースリポジトリを持つプロジェクトの手順に従うことができます。プロジェクトの作成の一環として IAM ロールを選択、または IAM ロールがない場合は追加してください。このプロジェクトには CodeCatalystWorkflowDevelopmentRole-***spaceName***サービスロールを使用することをお勧めします。

既にプロジェクトがある場合は、「」にスキップできますプロジェクトのリポジトリの表示。

> **Note**
>
> Space 管理者または Power ユーザーロールを持つユーザーのみが、 でプロジェクトを作成できます CodeCatalyst。このロールがなく、このチュートリアルで作業するプロジェクトが必要な場合は、これらのロールのいずれかを持つユーザーにプロジェクトを作成して、作成したプロジェクトに追加してもらいます。詳細については、「ユーザーロールによるアクセス許可の付与」を参照してください。

ブループリントを使用してプロジェクトを作成するには

1. CodeCatalyst コンソールで、プロジェクトを作成するスペースに移動します。
2. スペースダッシュボードで、[プロジェクトの作成] を選択します。
3. 設計図 で開始 を選択します。

4. CodeCatalyst ブループリントまたはスペースブループリントタブからブループリントを選択し、次へ を選択します。
5. プロジェクトの名前 に、プロジェクトに割り当てる名前とそれに関連するリソース名を入力します。名前はスペース内で一意でなければなりません。
6. (オプション) デフォルトでは、ブループリントによって作成されたソースコードはリポジトリに CodeCatalyst 保存されます。または、ブループリントのソースコードをサードパーティーのリポジトリに保存することもできます。詳細については、「で拡張機能を使用してプロジェクトに機能を追加する CodeCatalyst」を参照してください。

   使用するサードパーティーのリポジトリプロバイダーに応じて、次のいずれかを実行します。

   - GitHub リポジトリ： GitHub アカウントを接続します。

     詳細ドロップダウンメニューを選択し、リポジトリプロバイダーとして を選択し GitHub、設計図によって作成されたソースコードを保存する GitHub アカウントを選択します。

     > **Note**
     >
     > GitHub アカウントを接続する場合は、アイデンティティと CodeCatalyst アイデンティティの間にアイデンティティマッピングを確立するための個人用接続を作成する必要があります GitHub 。詳細については、「個人用接続」および「個人接続による GitHub リソースへのアクセス」を参照してください。

   - Bitbucket リポジトリ : Bitbucket ワークスペースを接続します。

     詳細ドロップダウンメニューを選択し、リポジトリプロバイダーとして Bitbucket を選択し、設計図によって作成されたソースコードを保存する Bitbucket ワークスペースを選択します。
7. プロジェクトリソース で、設計図パラメータを設定します。設計図によっては、ソースリポジトリ名に名前を付けるオプションがある場合があります。
8. (オプション) 行ったプロジェクトパラメータの選択に基づいて更新を含む定義ファイルを表示するには、「プロジェクトプレビューの生成」から「コードを表示」または「ワークフローを表示」を選択します。
9. (オプション) 設計図のカードから詳細を表示 を選択すると、設計図のアーキテクチャの概要、必要な接続とアクセス許可、設計図が作成するリソースの種類など、設計図に関する特定の詳細が表示されます。
10. [プロジェクトを作成] を選択します。

プロジェクト概要ページは、プロジェクトを作成するか、プロジェクトへの招待を承諾してサインインプロセスを完了するとすぐに開きます。新しいプロジェクトのプロジェクト概要ページには、未解決の問題やプルリクエストはありません。オプションで、問題を作成して自分に割り当てることもできます。また、他のユーザーをプロジェクトに招待することもできます。詳細については、「での問題の作成 CodeCatalyst」および「プロジェクトへのユーザーの招待」を参照してください。

プロジェクトのリポジトリの表示

プロジェクトのメンバーとして、プロジェクトのソースリポジトリを表示できます。追加のリポジトリを作成することもできます。Space 管理者ロールを持つユーザーがGitHub リポジトリまたは Bitbucket 拡張機能をインストールして設定している場合は、拡張機能用に設定された GitHub アカウントまたは Bitbucket ワークスペース内のサードパーティーリポジトリへのリンクを追加することもできます。詳細については、「ソースリポジトリの作成」および「クイックスタート: 拡張機能のインストール、プロバイダーの接続、 でのリソースのリンク CodeCatalyst」を参照してください。

> **Note**
>
> 単一ページのアプリケーション設計図で作成されたプロジェクトの場合、サンプルコードを含むソースリポジトリのデフォルト名は ***spa-app*** です。

プロジェクトのソースリポジトリに移動するには

1. プロジェクトに移動し、次のいずれかを実行します。
   - プロジェクトの概要ページで、リストから目的のリポジトリを選択し、リポジトリの表示を選択します。
   - ナビゲーションペインで [コード] を選択してから、[ソースリポジトリ] を選択します。ソースリポジトリ で、リストからリポジトリの名前を選択します。リポジトリのリストをフィルタリングするには、フィルターバーにリポジトリ名の一部を入力します。
2. リポジトリのホームページで、リポジトリの内容と、プルリクエストやワークフローの数など、関連するリソースに関する情報を表示します。デフォルトでは、デフォルトのブランチの内容が表示されます。ドロップダウンリストから別のブランチを選択して、ビューを変更できます。

リポジトリの概要ページには、このリポジトリとそのファイルのブランチ用に設定されたワークフローとプルリクエストに関する情報が含まれています。プロジェクトを作成したばかりの場合、コードを構築、テスト、デプロイする最初のワークフローは、完了するまで数分かかるため、引き続き実行されます。関連するワークフローとそのステータスを表示するには、関連するワークフロー の下

にある番号を選択しますが、CI/CD のワークフローページが開きます。このチュートリアルでは、概要ページにとどまり、リポジトリ内のコードを調べます。README.md ファイルの内容は、このページのリポジトリファイルの下にある でレンダリングされます。ファイル には、デフォルトのブランチの内容が表示されます。ファイルビューを変更して、別のブランチの内容がある場合はそれを表示できます。.codecatalyst フォルダには、ワークフロー YAML ファイルなど、プロジェクトの他の部分に使用されるコードが含まれています。

フォルダの内容を表示するには、フォルダ名の横にある矢印を選択して展開します。例えば、 の横にある矢印を選択すると、そのフォルダに含まれる単一ページのウェブアプリケーションのファイルsrcが表示されます。ファイルの内容を表示するには、リストから選択します。これにより、ファイルの表示 が開き、複数のファイルの内容を参照できます。コンソールで 1 つのファイルを編集することもできますが、複数のファイルを編集するには、開発環境を作成します。

開発環境の作成

Amazon CodeCatalyst コンソールのソースリポジトリでファイルを追加および変更できます。ただし、複数のファイルやブランチを効果的に操作するには、開発環境を使用するか、ローカルコンピュータにリポジトリをクローンすることをお勧めします。このチュートリアルでは、 という名前のブランチを使用して AWS Cloud9 開発環境を作成しますdevelop。別のブランチ名を選択できますが、ブランチに という名前を付けるとdevelop、このチュートリアルの後半でプルリクエストを作成するときに、ワークフローが自動的に実行され、コードの構築とテストが行われます。

> **Tip**
>
> 開発環境を使用する代わりに、または使用に加えてリポジトリのクローンをローカルに作成する場合は、ローカルコンピュータに Git があるか、IDE に Git が含まれていることを確認してください。詳細については、「ソースリポジトリを操作するための のセットアップ」を参照してください。

新しいブランチを使用して開発環境を作成するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 開発環境を作成するプロジェクトに移動します。
3. プロジェクトのソースリポジトリのリストからリポジトリの名前を選択します。または、ナビゲーションペインでコード を選択し、ソースリポジトリ を選択し、開発環境を作成するリポジトリを選択します。

4. リポジトリのホームページで、開発環境の作成 を選択します。
5. ドロップダウンメニューからサポートされている IDE を選択します。詳細については、「開発環境でサポートされる統合開発環境」を参照してください。
6. クローンするリポジトリを選択し、[新しいブランチで作業する] を選択し、[ブランチ名] フィールドにブランチ名を入力し、[ブランチの作成元] ドロップダウンメニューから新しいブランチを作成するブランチを選択します。
7. オプションで、開発環境のエイリアスを追加します。
8. オプションで、[開発環境設定] 編集ボタンを選択して、開発環境のコンピューティング、ストレージ、またはタイムアウト設定を編集します。
9. [作成] を選択します。開発環境の作成中は、開発環境のステータス列に [開始中] と表示され、開発環境が作成されると、ステータス列に [実行中] と表示されます。選択した IDE に開発環境を含む新しいタブが開きます。コードを編集し、変更をコミットしてプッシュできます。

開発環境を作成したら、ファイルを編集し、変更をコミットして、変更を**test**ブランチにプッシュできます。このチュートリアルでは、 `src`フォルダ内の `App.tsx` ファイル内の<p>タグ間のコンテンツを編集して、ウェブページに表示されるテキストを変更します。変更をコミットしてプッシュし、 CodeCatalyst タブに戻ります。

AWS Cloud9 開発環境から変更を加えてプッシュするには

1. で AWS Cloud9、サイドナビゲーションメニューを展開してファイルを参照します。を展開し`src`、 を開きます`App.tsx`。
2. <p> タグ内のテキストを変更します。
3. ファイルを保存し、Git メニューを使用して変更をコミットしてプッシュします。または、ターミナルウィンドウで、 および git push コマンドを使用して変更をコミットgit commitしてプッシュします。

```
git commit -am "Making an example change"
git push
```

> **Tip**
>
> Git コマンドを正常に実行する前に、ターミナルのディレクトリを Git リポジトリディレクトリに変更する必要がある場合があります。

プルリクエストの作成

プルリクエストを使用して、リリースされたソフトウェアのマイナーな変更や修正、主要な機能の追加、または新しいバージョンについて、コードの変更を共同で確認できます。このチュートリアルでは、###ブランチに加えた変更をメインブランチと比較して確認するためのプルリクエストを作成します。テンプレートを使用して作成されたプロジェクトでプルリクエストを作成すると、関連するワークフローの実行も開始されます。

プルリクエストを作成するには

1. プロジェクトに移動します。
2. 次のいずれかを行います。
   - ナビゲーションペインで、コード を選択し、プルリクエスト を選択し、プルリクエストの作成 を選択します。
   - リポジトリのホームページで、「その他」を選択し、「プルリクエストの作成」を選択します。
   - プロジェクトページで、プルリクエストの作成 を選択します。
3. ソースリポジトリ で、指定されたソースリポジトリがコミットされたコードを含むリポジトリであることを確認します。このオプションは、リポジトリのメインページからプルリクエストを作成していない場合にのみ表示されます。
4. 送信先ブランチ で、レビュー後にコードをマージするブランチを選択します。
5. ソースブランチ で、コミットされたコードを含むブランチを選択します。
6. プルリクエストのタイトル に、他のユーザーがレビューする必要がある内容と理由を理解するのに役立つタイトルを入力します。
7. (オプション) プルリクエストの説明 で、問題へのリンクや変更の説明などの情報を入力します。

> **Tip**
>
> プルリクエストに含まれる変更の説明 CodeCatalyst を自動的に生成するには、説明の書き込み を選択します。プルリクエストに追加した後、自動的に生成された説明を変更できます。
> この機能を使用するには、空間に対して生成 AI 機能を有効にする必要があります。詳細については、「生成 AI 機能の管理」を参照してください。

8. (オプション) 問題 で問題 をリンク を選択し、リストから問題を選択するか、その ID を入力します。問題のリンクを解除するには、リンク解除アイコンを選択します。
9. (オプション) 必要なレビューワー で、必要なレビューワーを追加 を選択します。プロジェクトメンバーのリストから選択して追加します。必要なレビューワーは、プルリクエストを送信先ブランチにマージする前に、変更を承認する必要があります。

> **Note**
>
> レビューワーを必須レビューワーとオプションのレビューワーの両方として追加することはできません。自分自身をレビューワーとして追加することはできません。

10. (オプション) オプションのレビューワー で、オプションのレビューワーを追加 を選択します。プロジェクトメンバーのリストから選択して追加します。オプションのレビューワーは、プルリクエストを送信先ブランチにマージする前に、変更を要件として承認する必要はありません。
11. ブランチの違いを確認します。プルリクエストに表示される違いは、ソースブランチのリビジョンと、プルリクエストの作成時の送信先ブランチのヘッドコミットであるマージベースとの間の変更です。変更が表示されない場合、ブランチは同じであるか、送信元と送信先の両方に同じブランチを選択している可能性があります。
12. プルリクエストにレビューするコードと変更が含まれていることを確認したら、「 の作成」を選択します。

> **Note**
>
> プルリクエストを作成したら、コメントを追加できます。コメントは、プルリクエスト、またはファイル内の個々の行、およびプルリクエスト全体に追加できます。ファイルなどのリソースへのリンクを追加するには、@ 記号の後にファイルの名前を付けます。

このプルリクエストの作成によって開始された関連するワークフローに関する情報を表示するには、概要を選択し、ワークフロー実行のプルリクエストの詳細エリアの情報を確認します。ワークフロー実行を表示するには、実行を選択します。

> **Tip**
>
> ブランチに 以外の名前を付けた場合**develop**、変更を構築してテストするためのワークフローは自動的に実行されません。これを設定する場合は、onPullRequestBuildAndTestワークフローの YAML ファイルを編集します。詳細については、「ワークフローの作成」を参照してください。

このプルリクエストにコメントし、他のプロジェクトメンバーにコメントを求めることができます。オプションまたは必要なレビューワーを追加または変更することもできます。リポジトリのソースブランチをさらに変更することを選択し、コミットされた変更によってプルリクエストのリビジョンがどのように作成されるかを確認できます。詳細については、、プルリクエストの確認、プルリクエストの更新、Amazon でのプルリクエストによるコードの確認 CodeCatalystおよび を参照してくださいワークフローの実行ステータスと詳細の表示。

プルリクエストのマージ

プルリクエストがレビューされ、必要なレビューワーから承認を受け取ったら、コンソールで CodeCatalystソースブランチを送信先ブランチにマージできます。プルリクエストをマージすると、送信先ブランチに関連付けられたワークフローを通じて変更の実行も開始されます。このチュートリアルでは、テストブランチを main にマージし、onPushToMainDeployPipelineワークフローの実行を開始します。

プルリクエストをマージするには (コンソール)

1. プルリクエスト で、前のステップで作成したプルリクエストを選択します。プルリクエストで、マージ を選択します。
2. プルリクエストに使用できるマージ戦略から選択します。オプションで、プルリクエストをマージした後にソースブランチを削除するオプションを選択または選択解除し、マージ を選択します。マージが完了すると、プルリクエストのステータスが Merged に変わり、プルリクエストのデフォルトビューに表示されなくなります。デフォルトのビューには、ステータスが Open のプルリクエストが表示されます。マージされたプルリクエストは引き続き表示できますが、承認やステータスの変更はできません。

> **Note**
>
> マージボタンがアクティブでない場合、またはマージ不可ラベルが表示される場合は、必要なレビューワーがプルリクエストをまだ承認していないか、 CodeCatalyst コ

ンソールでプルリクエストをマージできません。プルリクエストを承認していないレビューワーは、プルリクエストの詳細エリアの概要のクロックアイコンで示されます。必要なレビューワーがすべてプルリクエストを承認したが、マージボタンがまだアクティブでない場合、マージの競合が発生したり、スペースのストレージクォータを超えたりする可能性があります。開発環境の送信先ブランチのマージ競合を解決し、変更をプッシュしてからプルリクエストをマージするか、競合を解決してローカルにマージしてから、マージを含むコミットを にプッシュできます CodeCatalyst。詳細については、プルリクエストのマージ (Git)「」および Git ドキュメントを参照してください。

デプロイされたコードの表示

今度は、デフォルトブランチに最初にデプロイされたコードと、自動的に構築、テスト、デプロイされたマージされた変更を表示します。そのためには、リポジトリの概要ページに戻り、関連するワークフローアイコンの横にある番号を選択するか、ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。

デプロイされたコードを表示するには

1. ワークフロー で、 でonPushToMainDeployPipeline、最近の実行 を展開します。

   > **Note**
   >
   > これは、単一ページのアプリケーション設計図で作成されたプロジェクトのワークフローのデフォルト名です。

2. 最新の実行は、マージされたプルリクエストのコミットによってmainブランチに開始された実行であり、ステータスは「進行中」と表示される可能性があります。リストから正常に完了した実行を選択して、その実行の詳細を開きます。
3. 変数 を選択します。AppURLの値をコピーします。これは、デプロイされた単一ページのウェブアプリケーションの URL です。新しいブラウザタブを開き、 値に貼り付けて、ビルドおよびデプロイされたコードを表示します。タブは開いたままにします。
4. ワークフロー実行のリストに戻り、最新の実行が完了するまで待ちます。その場合は、開いたタブに戻り、ウェブアプリケーションを表示し、ブラウザを更新します。マージされたプルリクエストで行った変更が表示されます。

リソースのクリーンアップ

ソースリポジトリとプルリクエストの使用を検討したら、不要なリソースを削除できます。プルリクエストは削除できませんが、閉じることができます。作成したブランチはすべて削除できます。

ソースリポジトリやプロジェクトが不要になった場合は、それらのリソースを削除することもできます。詳細については、「ソースリポジトリの削除」および「プロジェクトの削除」を参照してください。

のプロジェクトのリポジトリにソースコードを保存する CodeCatalyst

ソースリポジトリは、プロジェクトのコードとファイルを安全に保存する場所です。また、最初のコミットから最新の変更までのソース履歴も保存されます。ソースリポジトリを含むブループリントを選択した場合、そのリポジトリには、プロジェクトのワークフローと通知の設定ファイルとその他の情報も含まれます。この設定情報は、.codecatalyst という名前のフォルダに保存されます。

でソースリポジトリを作成するには、プロジェクトの作成の一環としてソースリポジトリを作成するブループリントを持つプロジェクト CodeCatalyst を作成するか、既存のプロジェクトにソースリポジトリを作成します。プロジェクトユーザーは、プロジェクト用に作成したリポジトリを自動的に表示し、使用できるようになります。 GitHub または Bibucket でホストされている Git リポジトリをプロジェクトにリンクすることもできます。これを行うと、プロジェクトユーザーはプロジェクトのリポジトリのリストで、そのリンクされたリポジトリを表示してアクセスできます。

> **Note**
>
> リポジトリをリンクする前に、リポジトリをホストするサービスの 拡張機能をインストールする必要があります。アーカイブされたリポジトリをリンクすることはできません。空のリポジトリはリンクできますが、デフォルトのブランチを作成する最初のコミットで初期化 CodeCatalyst するまで で使用することはできません。詳細については、「スペースへの拡張機能のインストール」を参照してください。

デフォルトでは、ソースリポジトリは Amazon CodeCatalyst プロジェクトの他のメンバーと共有されます。プロジェクト用の追加のソースリポジトリを作成したり、リポジトリをプロジェクトにリンクしたりできます。プロジェクトのすべてのメンバーは、プロジェクトのソースリポジトリ内のファイルとフォルダを表示、追加、編集、削除できます。

ソースリポジトリ内のコードをすばやく操作するには、指定されたリポジトリをクローンして分岐する開発環境を作成し、開発環境用に選択した統合開発環境 (IDE) でコードを操作することができます。ローカルコンピュータでソースリポジトリのクローンを作成し、ローカルリポジトリと のリモートリポジトリとの間で変更をプルおよびプッシュできます CodeCatalyst。ソースリポジトリを操作するには、 IDE が認証情報管理をサポートしている限り、任意の IDE でソースリポジトリへのアクセスを設定します。

リポジトリ名は、 CodeCatalyst プロジェクト内で一意である必要があります。

トピック

ソースリポジトリの作成

Amazon のブループリントを使用してプロジェクトを作成すると CodeCatalyst、 によってソースリポジトリ CodeCatalyst が作成されます。そのソースリポジトリには、ワークフローやその他のリソースの設定情報に加えて、サンプルコードが含まれています。これは、 でリポジトリの使用を開始する推奨方法です CodeCatalyst。プロジェクトのリポジトリを作成することもできます。これらのリポジトリには 1 つのファイル、つまりいつでも編集または削除できる README.md ファイルが含まれます。ソースリポジトリの作成時に選択した内容によっては、リポジトリに `.gitignore` ファイルが含まれている場合もあります。

ソースリポジトリを作成するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトに移動します。
3. ナビゲーションペインで [コード] を選択してから、[ソースリポジトリ] を選択します。
4. [リポジトリの追加] を選択し、[リポジトリの作成] を選択します。
5. [リポジトリ名] で、リポジトリの名前を指定します。このガイドでは を使用していますが*`codecatalyst-source-repository`*、別の名前を選択できます。リポジトリ名は、プロ

ジェクト内で一意である必要があります。リポジトリ名の要件の詳細については、「」を参照してくださいのソースリポジトリのクォータ CodeCatalyst。

6. (オプション) 説明 で、プロジェクトの他のユーザーがリポジトリが何に使用されるかを理解するのに役立つリポジトリの説明を追加します。
7. (オプション) プッシュする予定のコードタイプの.gitignoreファイルを追加します。
8. [作成] を選択します。

> **Note**
>
> CodeCatalyst は、作成時にリポジトリにREADME.mdファイルを追加します。CodeCatalyst は、main という名前のデフォルトブランチにリポジトリの初期コミットも作成します。README.md ファイルは編集または削除できますが、デフォルトブランチを変更または削除することはできません。

ソースリポジトリのリンク

ソースリポジトリをプロジェクトにリンクする場合、リポジトリをホストするサービスの CodeCatalyst 拡張子を持つリポジトリを、その拡張子がスペースにインストールされている場合は含めることができます。Space 管理者ロールを持つユーザーのみが拡張機能をインストールできます。拡張機能をインストールしたら、その拡張機能によってアクセス用に設定されたリポジトリにリンクできます。詳細については、スペースへの拡張機能のインストール「」または「」を参照してくださいでの GitHub リポジトリ、Bitbucket リポジトリ、Jira プロジェクトのリンク CodeCatalyst。

> **Important**
>
> GitHub または Bitbucket リポジトリを寄稿者 としてリンクすることはできますが、サードパーティーリポジトリのリンクを解除できるのは、スペース管理者またはプロジェクト管理者 のみです。詳細については、「での GitHub リポジトリ、Bitbucket リポジトリ、Jira プロジェクトのリンク解除 CodeCatalyst」を参照してください。

> **Important**
>
> リポジトリ拡張をインストールすると、リンク先のリポジトリのコード CodeCatalyst のインデックスが作成され、 に保存されます CodeCatalyst。これにより、コードは で検索できるようになります CodeCatalyst。でリンクされたリポジトリを使用する際のコードのデータ

保護について理解を深めるには CodeCatalyst、「Amazon CodeCatalyst ユーザーガイド」の「データ保護」を参照してください。

> **Note**
>
> - GitHub または Bitbucket リポジトリは、スペース内の 1 つの CodeCatalyst プロジェクトにのみリンクできます。
> - 空のリポジトリ、アーカイブされたリポジトリ GitHub 、または Bitbucket リポジトリを CodeCatalyst プロジェクトで使用することはできません。
> - プロジェクト内の CodeCatalyst リポジトリと同じ名前の GitHub または Bitbucket リポジトリをリンクすることはできません。
> - GitHub リポジトリ拡張機能は GitHub Enterprise Server リポジトリと互換性がありません。
> - Bitbucket リポジトリ拡張機能は Bitbucket データセンターリポジトリと互換性がありません。

ソースリポジトリをリンクするには

1. リポジトリをリンクするプロジェクトに移動します。

   > **Note**
   >
   > リポジトリをリンクする前に、Space 管理者ロールを持つユーザーは、まずリポジトリをホストするプロバイダーの拡張機能をインストールする必要があります。詳細については、「スペースへの拡張機能のインストール」を参照してください。

2. ナビゲーションペインで [コード] を選択してから、[ソースリポジトリ] を選択します。
3. リポジトリの追加 を選択し、リポジトリのリンク を選択します。
4. リポジトリプロバイダードロップダウンメニューから、サードパーティーのリポジトリプロバイダー GitHubまたは Bitbucket のいずれかを選択します。
5. リンクを選択したサードパーティーのリポジトリプロバイダーに応じて、次のいずれかを実行します。

   - GitHub リポジトリ: GitHub リポジトリをリンクします。

1. GitHub アカウントのドロップダウンメニューから、リンクするリポジトリを含む GitHub アカウントを選択します。
2. GitHub リポジトリのドロップダウンメニューから、 CodeCatalyst プロジェクトをリンクする GitHub アカウントを選択します。
3. （オプション) GitHub リポジトリのリストにリポジトリが表示されない場合は、 の Amazon CodeCatalyst アプリケーションでリポジトリアクセスが設定されていない可能性があります GitHub。接続されたアカウントの CodeCatalyst で使用できる GitHub リポジトリを設定できます。

   a. GitHub アカウントに移動し、設定 を選択し、アプリケーション を選択します。

   b. Installed GitHub Apps タブで、Amazon CodeCatalyst アプリケーションの設定を選択します。

   c. でリンクする GitHub リポジトリへのアクセスを設定するには、次のいずれかを実行します CodeCatalyst。

      - 現在および将来のすべてのリポジトリへのアクセスを提供するには、すべてのリポジトリ を選択します。
      - 特定のリポジトリへのアクセスを許可するには、リポジトリの選択のみ を選択し、リポジトリの選択 ドロップダウンを選択し、 でリンクすることを許可するリポジトリを選択します CodeCatalyst。

- Bitbucket リポジトリ : Bitbucket リポジトリをリンクします。

1. Bitbucket ワークスペースのドロップダウンメニューから、リンクするリポジトリを含む Bitbucket ワークスペースを選択します。
2. Bitbucket リポジトリのドロップダウンメニューから、 CodeCatalyst プロジェクトをリンクする Bitbucket リポジトリを選択します。

> **Tip**
>
> リポジトリの名前がグレー表示になっている場合、そのリポジトリは Amazon 内の別のプロジェクトに既にリンクされているため、リンクできません CodeCatalyst。

6. [Link (リンク)] を選択します。

で GitHub または Bitbucket リポジトリを使用しなくなった場合は CodeCatalyst、プロジェクトから CodeCatalyst リンクを解除できます。リポジトリのリンクが解除されると、そのリポジトリのイベントはワークフロー実行を開始せず、 CodeCatalyst 開発環境でそのリポジトリを使用することはできません。詳細については、「での GitHub リポジトリ、Bitbucket リポジトリ、Jira プロジェクトのリンク解除 CodeCatalyst」を参照してください。

ソースリポジトリの表示

Amazon のプロジェクトに関連付けられたソースリポジトリを表示できます CodeCatalyst。のソースリポジトリの場合 CodeCatalyst、リポジトリの概要ページには、そのリポジトリ内の情報とアクティビティの概要が次のように表示されます。

- リポジトリがある場合は、その説明
- リポジトリ内のブランチの数
- リポジトリのオープンプルリクエストの数
- リポジトリの関連ワークフローの数
- デフォルトブランチのファイルとフォルダ、または選択したブランチ
- 表示されたブランチへの最後のコミットのタイトル、作成者、日付
- README.md ファイルが含まれている場合、Markdown でレンダリングされた README.md ファイルの内容

このページには、リポジトリのコミット、ブランチ、プルリクエストへのリンクと、個々のファイルをすばやく開いたり、表示したり、編集したりする方法も用意されています。

> **Note**
>
> リンクされたリポジトリに関するこの情報は、 CodeCatalyst コンソールでは表示できません。リンクされたリポジトリに関する情報を表示するには、リポジトリのリストにあるリンクを選択して、リポジトリをホストするサービスでそのリポジトリを開きます。

プロジェクトのソースリポジトリに移動するには

1. プロジェクトに移動し、次のいずれかを実行します。
   - プロジェクトの概要ページで、リストから目的のリポジトリを選択し、リポジトリの表示を選択します。

- ナビゲーションペインで [コード] を選択してから、[ソースリポジトリ] を選択します。ソースリポジトリ で、リストからリポジトリの名前を選択します。リポジトリのリストをフィルタリングするには、フィルターバーにリポジトリ名の一部を入力します。

2. リポジトリのホームページで、リポジトリの内容と、プルリクエストやワークフローの数など、関連するリソースに関する情報を表示します。デフォルトでは、デフォルトのブランチの内容が表示されます。ドロップダウンリストから別のブランチを選択して、ビューを変更できます。

> **Tip**
>
> プロジェクトの概要ページからプロジェクトコードを表示を選択して、プロジェクトのリポジトリにすばやく移動することもできます。

ソースリポジトリの設定の編集

リポジトリの説明の編集、デフォルトのブランチの選択、ブランチルールの作成と管理、 でのプルリクエストの承認ルールの作成と管理など、リポジトリの設定を管理できます CodeCatalyst。これにより、プロジェクトメンバーがリポジトリが何に使用されているかを理解し、チームが使用するベストプラクティスとプロセスを適用できます。

> **Note**
>
> ソースリポジトリの名前は編集できません。
> でリンクされたリポジトリの名前、説明、その他の情報を編集することはできません CodeCatalyst。リンクされたリポジトリに関する情報を変更するには、リンクされたリポジトリをホストするプロバイダーでそのリポジトリを編集する必要があります。詳細については、リンクされたリポジトリをホストするサービスのドキュメントを参照してください。

リポジトリの設定を編集するには

1. CodeCatalyst コンソールで、設定を編集するソースリポジトリを含むプロジェクトに移動します。
2. プロジェクトの概要ページで、リストから目的のリポジトリを選択し、リポジトリの表示 を選択します。または、ナビゲーションペインでコード を選択し、ソースリポジトリ を選択します。プロジェクトのソースリポジトリのリストからリポジトリの名前を選択します。

3. リポジトリの概要ページで、詳細 を選択し、設定の管理 を選択します。
4. 次の 1 つ以上の操作を行います。

   - リポジトリの説明を編集し、保存 を選択します。
   - リポジトリのデフォルトブランチを変更するには、デフォルトブランチ で、編集 を選択します。詳細については、「リポジトリのデフォルトブランチの管理」を参照してください。
   - ブランチで特定のアクションを実行するアクセス許可を持つプロジェクトロールのルールを追加、削除、または変更するには、ブランチルール で の編集 を選択します。詳細については、「ブランチルールを使用してブランチに対して許可されるアクションを管理する」を参照してください。
   - プルリクエストをブランチにマージするための承認ルールを追加、削除、または変更するには、承認ルール で の編集 を選択します。詳細については、「プルリクエストを承認ルールとマージするための要件の管理」を参照してください。

ソースリポジトリのクローン作成

ソースリポジトリで複数のファイル、ブランチ、コミットを効果的に操作するには、ソースリポジトリをローカルコンピュータにクローンし、Git クライアントまたは統合開発環境 (IDE) を使用して変更を行います。問題やプルリクエストなどの CodeCatalyst 機能を操作するために、変更をコミットしてソースリポジトリにプッシュします。コードを操作する開発環境を作成することもできます。開発環境を作成すると、指定したリポジトリとブランチが開発環境に自動的にクローンされます。

> **Note**
>
> リンクされたリポジトリを CodeCatalyst コンソールでクローンしたり、開発環境を作成したりすることはできません。リンクされたリポジトリをローカルでクローンするには、リポジトリのリストにあるリンクを選択して、リポジトリをホストするサービスでそのリポジトリを開き、クローンを作成します。詳細については、リンクされたリポジトリをホストするサービスのドキュメントを参照してください。

ソースリポジトリから開発環境を作成するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. ナビゲーションペインで [コード] を選択してから、[ソースリポジトリ] を選択します。
3. コードを操作するソースリポジトリを選択します。

4. [開発環境を作成] を選択します。
5. ドロップダウンメニューからサポートされている IDE を選択します。詳細については、「開発環境でサポートされる統合開発環境」を参照してください。
6. 次のいずれかを行います。
   - 「既存のブランチで作業」を選択し、「既存のブランチ」ドロップダウンメニューからブランチを選択します。
   - 新しいブランチで作業 を選択し、ブランチ名フィールドにブランチ名を入力し、ドロップダウンメニューから新しいブランチを作成するブランチを選択します。
7. 必要に応じて、開発環境の名前を追加するか、設定を編集します。
8. [作成] を選択します。

ソースリポジトリのクローンを作成するには

1. プロジェクトに移動します。
2. プロジェクトの概要ページで、リストから目的のリポジトリを選択し、リポジトリの表示 を選択します。または、ナビゲーションペインでコード を選択し、ソースリポジトリ を選択します。プロジェクトのソースリポジトリのリストからリポジトリの名前を選択します。リポジトリのリストをフィルタリングするには、フィルターバーにリポジトリ名の一部を入力します。
3.
4. リポジトリ のクローンを作成するを選択します。リポジトリのクローン URL をコピーします。

> **Note**
>
> 個人用アクセストークン (PAT) がない場合は、トークンの作成 を選択します。トークンをコピーして安全な場所に保存します。この PAT は、Git クライアントまたは統合開発環境 (IDE) によってパスワードの入力を求められたときに使用します。

5. 次のいずれかを行います。
   - ローカルコンピュータにリポジトリのクローンを作成するには、ターミナルまたはコマンドラインを開き、git cloneコマンドの後にクローン URL を指定してコマンドを実行します。例:

```
git clone https://LiJuan@git.us-west-2.codecatalyst.aws/
v1/ExampleCorp/MyExampleProject/MyExampleRepo
```

パスワードの入力を求められたら、前に保存した PAT を貼り付けます。

> **Note**
>
> オペレーティングシステムが認証情報管理を提供している場合、または認証情報管理システムをインストールしている場合は、PAT を 1 回だけ指定する必要があります。そうでない場合は、Git オペレーションごとに PAT を指定する必要があります。ベストプラクティスとして、認証情報管理システムが PAT を安全に保存していることを確認してください。クローン URL 文字列の一部として PAT を含めないでください。

- IDE を使用してリポジトリのクローンを作成するには、IDE のドキュメントに従ってください。Git リポジトリのクローンを作成し、URL を指定するオプションを選択します。パスワードの入力を求められたら、PAT を指定します。

ソースリポジトリの削除

Amazon CodeCatalyst プロジェクトのソースリポジトリが不要になった場合は、削除できます。ソースリポジトリを削除すると、リポジトリに保存されているプロジェクト情報もすべて削除されます。ワークフローがソースリポジトリに依存している場合、それらのワークフローは、リポジトリが削除された後にプロジェクトワークフローのリストから削除されます。ソースリポジトリを参照する問題は削除または変更されませんが、問題に追加されたソースリポジトリへのリンクは、リポジトリが削除されると失敗します。

> **Important**
>
> ソースリポジトリの削除は元に戻すことができません。ソースリポジトリを削除すると、そのリポジトリのクローンを作成したり、そこからデータを取得したり、そのリポジトリにデータをプッシュしたりできなくなります。ソースリポジトリを削除しても、そのリポジトリのローカルコピー (ローカルリポジトリ) は削除されません。ローカルリポジトリを削除するには、ローカルコンピュータのディレクトリとファイル管理ツールを使用します。

> **Note**
>
> リンクされたリポジトリは CodeCatalyst コンソールで削除できません。リンクされたリポジトリを削除するには、リポジトリのリストにあるリンクを選択して、リポジトリをホスト

するサービスでそのリポジトリを開き、削除します。詳細については、リンクされたリポジトリをホストするサービスのドキュメントを参照してください。
リンクされたリポジトリをプロジェクトから削除するには、「」を参照してくださいでの GitHub リポジトリ、Bitbucket リポジトリ、Jira プロジェクトのリンク解除 CodeCatalyst。

ソースリポジトリを削除するには

1. 削除するソースリポジトリを含むプロジェクトに移動します。
2. プロジェクトの概要ページで、リストから目的のリポジトリを選択し、リポジトリの表示 を選択します。または、ナビゲーションペインでコード を選択し、ソースリポジトリ を選択します。プロジェクトのソースリポジトリのリストからリポジトリの名前を選択します。
3. リポジトリのホームページで、「その他」を選択し、「リポジトリの削除」を選択します。
4. ブランチ、プルリクエスト、および関連するワークフロー情報を確認して、まだ使用中または未完了の作業があるリポジトリを削除していないことを確認します。続行する場合は、「削除」と入力し、「削除」を選択します。

ソースコードを整理して Amazon のブランチを操作する CodeCatalyst

Git では、ブランチはコミットへのポインターまたは参照です。開発時、ブランチはタスクを整理するのに便利な方法です。ブランチを使用すると、他のブランチの作業に影響を与えることなく、新しいバージョンまたは異なるバージョンのファイルで個別に作業を行うことができます。ブランチを使用して、新機能の開発、プロジェクトの特定のバージョンの保存などを行うことができます。ソースリポジトリ内のブランチのルールを設定して、ブランチに対する特定のアクションをそのプロジェクトの特定のロールに制限できます。

Amazon のソースリポジトリ CodeCatalyst には、作成方法に関係なく、コンテンツとデフォルトのブランチがあります。リンクされたリポジトリにはデフォルトのブランチやコンテンツがない場合がありますが、初期化してデフォルトのブランチを作成する CodeCatalyst まで では使用できません。設計図を使用してプロジェクトを作成すると、 は README.md ファイル、サンプルコード、ワークフロー定義、その他のリソースを含むそのプロジェクトのソースリポジトリ CodeCatalyst を作成します。ブループリントを使用せずにソースリポジトリを作成すると、README.md ファイルが最初のコミットとして追加され、デフォルトのブランチが作成されます。このデフォルトのブランチは main という名前です。このデフォルトブランチは、ユーザーがリポジトリをクローンするときに、ローカルリポジトリのベースブランチまたはデフォルトブランチとして使用されるブランチです。

> **Note**
>
> デフォルトのブランチは削除できません。ソースリポジトリ用に作成された最初のブランチは、そのリポジトリのデフォルトブランチです。さらに、検索ではデフォルトのブランチの結果のみが表示されます。他のブランチでコードを検索することはできません。

でリポジトリを作成すると、最初のコミット CodeCatalyst も作成され、 README.md ファイルを含むデフォルトのブランチが作成されます。そのデフォルトブランチの名前はメイン です。これは、このガイドの例で使用されているデフォルトのブランチ名です。

トピック

ブランチの作成

CodeCatalyst コンソールを使用してリポジトリにブランチを作成できます CodeCatalyst 。作成したブランチは、他のユーザーが次にリポジトリから変更をプルしたときに表示されます。

> **Tip**
>
> コードを操作する開発環境の作成の一環としてブランチを作成することもできます。詳細については、「開発環境の作成」を参照してください。

Git を使用してブランチを作成することもできます。詳細については、「ブランチの一般的な Git コマンド」を参照してください。

ブランチを作成するには (コンソール)

1. CodeCatalyst コンソールで、ソースリポジトリが存在するプロジェクトに移動します。

2. プロジェクトのソースリポジトリのリストからリポジトリの名前を選択します。または、ナビゲーションペインでコード を選択し、ソースリポジトリ を選択します。
3. ブランチを作成するリポジトリを選択します。
4. リポジトリの概要ページで、「詳細」を選択し、「ブランチの作成」を選択します。
5. ブランチの名前を入力します。
6. ブランチを作成するブランチを選択し、 の作成を選択します。

リポジトリのデフォルトブランチの管理

Amazon のソースリポジトリのデフォルトブランチとして使用するブランチを指定できます CodeCatalyst。のすべてのソースリポジトリ CodeCatalyst には、作成方法に関係なく、コンテンツとデフォルトのブランチがあります。設計図を使用してプロジェクトを作成する場合、そのプロジェクト用に作成されたソースリポジトリのデフォルトブランチには main という名前が付けられます。デフォルトのブランチの内容は、そのリポジトリの概要ページに自動的に表示されます。

デフォルトのブランチは、ソースリポジトリ内の他のすべてのブランチとは少し異なります。名前の横に特別なラベル、デフォルト があります。デフォルトブランチは、ユーザーが Git クライアントを使用してローカルコンピュータにリポジトリのクローンを作成するときに、ローカルリポジトリ (リポジトリ) のベースブランチまたはデフォルトブランチとして使用されるブランチです。また、ワークフロー YAML ファイルを保存したり、問題に関する情報を保存したりするためのワークフローを作成するときにも使用されるデフォルトです。で検索を使用する場合 CodeCatalyst、リポジトリのデフォルトブランチのみが検索されます。デフォルトのブランチはプロジェクトの非常に多くの側面の基礎であるため、デフォルトのブランチとして指定されているブランチは削除できません。ただし、デフォルトのブランチとして別のブランチを使用することを選択できます。これを行うと、以前のデフォルトブランチに適用されたブランチルールは、デフォルトブランチとして指定したブランチに自動的に適用されます。

> **Note**
>
> プロジェクトのソースリポジトリのデフォルトブランチを変更するには、 CodeCatalyst プロジェクト管理者ロールが必要です。これは、リンクされたリポジトリには適用されません。

リポジトリのデフォルトブランチを表示および変更するには

1. リポジトリが存在するプロジェクトに移動します。

2. プロジェクトのソースリポジトリのリストからリポジトリの名前を選択します。または、ナビゲーションペインでコード を選択し、ソースリポジトリ を選択します。

   デフォルトのブランチを含む、設定を表示するリポジトリを選択します。
3. リポジトリの概要ページで、詳細 を選択し、設定の管理 を選択します。
4. デフォルトブランチ では、デフォルトブランチとして指定されたブランチの名前が、名前の横にデフォルトというラベルとともに表示されます。この同じラベルは、ブランチ のブランチのリストのブランチ名の横に表示されます。
5. デフォルトのブランチを変更するには、編集 を選択します。

   > **Note**
   >
   > デフォルトのブランチを変更するには、プロジェクトにプロジェクト管理者ロールが必要です。

6. ドロップダウンリストからデフォルトのブランチを作成するブランチの名前を選択し、保存 を選択します。

ブランチルールを使用してブランチに対して許可されるアクションを管理する

ブランチを作成すると、そのロールのアクセス許可に基づいて、そのブランチに対して特定のアクションが許可されます。ブランチルールを設定することで、特定のブランチに対して許可されるアクションを変更できます。ブランチルールは、ユーザーがプロジェクトで持っているロールに基づいています。コミットをブランチにプッシュするなど、いくつかの事前定義されたアクションを、プロジェクト内の特定のロールを持つユーザーに制限できます。これにより、特定のアクションを実行できるロールを制限することで、プロジェクト内の特定のブランチを保護するのに役立ちます。例えば、プロジェクト管理者ロールを持つユーザーのみがそのブランチにマージまたはプッシュできるようにブランチルールを設定すると、プロジェクト内の他のロールを持つユーザーは、そのブランチのコードを変更できなくなります。

ブランチのルールを作成する場合の影響をすべて慎重に検討する必要があります。例えば、ブランチへのプッシュをプロジェクト管理者ロールを持つユーザーに制限すると、コントリビューターロールを持つユーザーはそのブランチでワークフローを作成または編集できなくなります。ワークフロー YAML はそのブランチに保存され、これらのユーザーは変更をコミットして YAML にプッシュできないためです。ベストプラクティスとして、ブランチルールを作成した後でテストして、意図していない影響がないことを確認します。ブランチルールをプルリクエストの承認ルールと組み合わせて使

用することもできます。詳細については、「プルリクエストを承認ルールとマージするための要件の管理」を参照してください。

> **Note**
>
> プロジェクト内のソースリポジトリのブランチルールを管理するには、 CodeCatalyst プロジェクト管理者ロールが必要です。リンクされたリポジトリのブランチルールを作成することはできません。
> ロールのデフォルトのアクセス許可よりも制限の厳しいブランチルールのみを作成できます。プロジェクト内のユーザーのロールで許可されているよりも許容度の高いブランチルールを作成することはできません。例えば、レビューワーロールを持つユーザーがブランチにプッシュできるようにするブランチルールを作成することはできません。

ソースリポジトリのデフォルトブランチに適用されるブランチルールは、他のブランチに適用されるブランチルールとは少し動作が異なります。デフォルトブランチに適用されるルールは、デフォルトブランチとして指定したブランチに自動的に適用されます。以前はデフォルトブランチとして設定されていたブランチは、削除に対する保護がなくなる点を除いて、そのブランチに適用されたルールを保持します。この保護は、現在のデフォルトブランチにのみ適用されます。

ブランチルールには、標準 とカスタム の 2 つの状態があります。Standard は、ブランチで許可されるアクションが、ユーザーが CodeCatalyst ブランチアクションに対して持つロールのアクセス許可と一致するアクションであることを示します。どのロールにどのアクセス許可があるかについては、「」を参照してくださいユーザーロールによるアクセス許可の付与。カスタム は、1 つ以上のブランチアクションに、プロジェクト内のユーザーのロールによって付与されたデフォルトのアクセス許可とは異なる、そのアクションを実行できる特定のロールのリストを持つアクションがあることを示します。

> **Note**
>
> ブランチの 1 つ以上のアクションを制限するブランチルールを作成する場合、ブランチの削除アクションは、プロジェクト管理者ロールを持つユーザーのみがそのブランチを削除できるように自動的に設定されます。

次の表に、ブランチでこれらのアクションを実行できるロールのアクションとデフォルト設定を示します。

ブランチアクションとロール

ブランチアクション	ブランチルールが適用されない場合にこのアクションを実行できるロール
ブランチにマージする (これには、プルリクエストをブランチにマージすることが含まれます)	プロジェクト管理者、寄稿者
ブランチにプッシュする	プロジェクト管理者、寄稿者
ブランチを削除する	プロジェクト管理者、寄稿者
ブランチを削除する (デフォルトブランチ)	許可されていません

ブランチルールは削除できませんが、ブランチでこのアクションを実行できるすべてのロールからのアクションを許可するように更新できます。これにより、ルールは事実上削除されます。

> **Note**
>
> プロジェクト内のソースリポジトリのブランチルールを設定するには、 CodeCatalyst プロジェクト管理者ロールが必要です。これは、リンクされたリポジトリには適用されません。リンクされたリポジトリは、 のブランチルールをサポートしていません CodeCatalyst。

リポジトリのブランチルールを表示および編集するには

1. リポジトリが存在するプロジェクトに移動します。
2. プロジェクトのソースリポジトリのリストからリポジトリの名前を選択します。または、ナビゲーションペインでコード を選択し、ソースリポジトリ を選択します。

   ブランチルールを表示するリポジトリを選択します。
3. リポジトリの概要ページで、ブランチ を選択します。
4. ブランチルール列で、リポジトリの各ブランチのルールのステータスを表示します。Standard は、ブランチアクションのルールがソースリポジトリで作成されたブランチのデフォルトルールであり、プロジェクト内のそれらのロールに付与されたアクセス許可と一致することを示します。カスタム は、1 つ以上のブランチアクションに、そのブランチで許可される 1 つ以上のアクションを別のロールセットに制限するルールがあることを示します。

ブランチのブランチルールの詳細を表示するには、確認するブランチの横にある「標準」または「カスタム」を選択します。

5. ブランチルールを作成または変更するには、設定の管理 を選択します。ソースリポジトリの設定ページのブランチルール で、編集 を選択します。
6. ブランチ で、ルールを設定するブランチの名前をドロップダウンリストから選択します。許可されるアクションタイプごとに、ドロップダウンリストからそのアクションを実行することを許可するロールを選択し、保存 を選択します。

ブランチの Git コマンド

Git を使用して、コンピュータ (ローカルリポジトリ) または開発環境にあるソースリポジトリのクローンでブランチを作成、管理、削除し、変更をコミットして CodeCatalyst ソースリポジトリ (リモートリポジトリ) にプッシュできます。例:

ブランチの一般的な Git コマンド

ローカルリポジトリ内のすべてのブランチを一覧表示します。現在のブランチの横にはアスタリスク (*) が表示されます。	`git branch`
リモートリポジトリ内のすべての既存のブランチに関する情報をローカルリポジトリにプルします。	`git fetch`
ローカルリポジトリ内のすべてのブランチと、ローカルリポジトリ内のリモート追跡ブランチを一覧表示します。	`git branch -a`
ローカルリポジトリ内のリモート追跡ブランチのみを一覧表示します。	`git branch -r`
指定されたブランチ名を使用して、ローカルリポジトリにブランチを作成します。このブランチは、変更をコミットしてプッシュするまでリモートリポジトリに表示されません。	`git branch branch-name`

指定されたブランチ名を使用してローカルリポジトリにブランチを作成し、そのブランチに切り替えます。	`git checkout -b branch-name`
指定したブランチ名を使用して、ローカルリポジトリ内で別のブランチに切り替えます。	`git checkout other-branch-name`
リモートリポジトリのローカルリポジトリの指定されたニックネームと指定されたブランチ名を使用して、ローカルリポジトリからリモートリポジトリにブランチをプッシュします。また、ローカルリポジトリのブランチのアップストリーム追跡情報を設定します。	`git push -u remote-name branch-name`
ローカルリポジトリ内の別のブランチからローカルリポジトリ内の現在のブランチに変更をマージします。	`git merge from-other-branch-name`
マージされていない作業が含まれていない限り、ローカルリポジトリ内のブランチを削除します。	`git branch -d branch-name`
ローカルリポジトリがリモートリポジトリに持つニックネームとブランチ名を指定して、リモートリポジトリ内のブランチを削除します。(コロン (:) の使用に注意してください)。または、 コマンド`--delete`の一部として を指定します。	`git push remote-name :branch-name` `git push remote-name --delete branch-name`

詳細については、Git のドキュメントを参照してください。

ブランチと詳細の表示

Amazon コンソールでは CodeCatalyst、特定のブランチのファイル、フォルダ、最新のコミットの詳細など、Amazon CodeCatalyst のリモートブランチに関する情報を表示できます。Git コマンドとローカルオペレーティングシステムを使用して、リモートブランチとローカルブランチのこの情報を表示することもできます。

ブランチを表示するには (コンソール)

1. CodeCatalyst コンソールで、ブランチを表示するソースリポジトリを含むプロジェクトに移動します。コード を選択し、ソースリポジトリ を選択し、ソースリポジトリを選択します。
2. プロジェクトのソースリポジトリのリストからリポジトリの名前を選択します。または、ナビゲーションペインでコード を選択し、ソースリポジトリ を選択します。

   ブランチを表示するリポジトリを選択します。
3. リポジトリのデフォルトブランチが表示されます。ブランチ内のファイルとフォルダのリスト、最新のコミットに関する情報、ブランチに存在する場合は README.md ファイルの内容を確認できます。別のブランチの情報を表示するには、リポジトリのブランチのドロップダウンリストから選択します。
4. リポジトリのすべてのブランチを表示するには、すべての を表示 を選択します。ブランチページには、各ブランチの名前、最新のコミット、およびルールに関する情報が表示されます。

Git とオペレーティングシステムを使用してブランチと詳細を表示する方法については、「ブランチの共通 Git コマンド」、「Git ドキュメント」、および「オペレーティングシステムのドキュメント」を参照してください。

ブランチの削除

ブランチが不要になった場合は、削除することができます。例えば、機能変更のあるブランチをデフォルトブランチにマージし、その機能がリリースされた場合、変更がすでにデフォルトブランチの一部であるため、元の機能ブランチを削除できます。ブランチの数を少なくしておくと、ユーザーが作業する変更を含むブランチを見つけるのに役立ちます。ブランチを削除すると、そのブランチのコピーは、ユーザーが変更をプルして同期するまで、ローカルコンピュータのリポジトリのクローンに残ります。

ブランチを削除するには (コンソール)

1. リポジトリが存在するプロジェクトに移動します。
2. プロジェクトのソースリポジトリのリストからリポジトリの名前を選択します。または、ナビゲーションペインでコード を選択し、ソースリポジトリ を選択します。

   ブランチを削除するリポジトリを選択します。
3. リポジトリの概要ページで、ブランチ名の横にあるドロップダウンセレクターを選択し、すべての を表示を選択します。

4. 削除するブランチを選択し、ブランチの削除 を選択します。

> **Note**
>
> リポジトリのデフォルトブランチは削除できません。

5. 確認のダイアログボックスが表示されます。リポジトリ、開いているプルリクエストの数、ブランチに関連付けられたワークフローの数が表示されます。
6. ブランチの削除を確認するには、テキストボックスに delete と入力し、Delete を選択します。

Git を使用してブランチを削除することもできます。詳細については、「ブランチの一般的な Git コマンド」を参照してください。

Amazon でのソースコードファイルの管理 CodeCatalyst

Amazon では CodeCatalyst、ファイルはバージョン管理された自己完結型の情報であり、ファイルが保存されているソースリポジトリとブランチの他のユーザーが利用できます。リポジトリファイルはディレクトリ構造で整理できます。 CodeCatalyst は、ファイルへのコミットされたすべての変更を自動的に追跡します。ファイルの異なるバージョンを異なるリポジトリブランチに保存できます。

ソースリポジトリに複数のファイルを追加または編集するには、Git クライアント、開発環境、または統合開発環境 (IDE) を使用できます。1 つのファイルを追加または編集するには、 CodeCatalyst コンソールを使用できます。

トピック

- ファイルの作成または追加
- ファイルの表示
- ファイルの変更履歴の表示
- ファイルの編集
- ファイルの名前変更または削除

ファイルの作成または追加

ソースリポジトリにファイルを作成して追加するには、Amazon CodeCatalyst コンソール、開発環境、接続された統合開発環境 (IDE)、または Git クライアントを使用できます。 CodeCatalyst コンソールには、ファイルを作成するためのコードエディタが含まれています。このエディタは、リポジ

トリのブランチで README.md ファイルなどのシンプルなファイルを作成または編集するのに便利です。複数のファイルを操作する場合は、開発環境 の作成を検討してください。

ソースリポジトリから開発環境を作成するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. ナビゲーションペインで [コード] を選択してから、[ソースリポジトリ] を選択します。
3. コードを操作するソースリポジトリを選択します。
4. [開発環境を作成] を選択します。
5. ドロップダウンメニューからサポートされている IDE を選択します。詳細については、「開発環境でサポートされる統合開発環境」を参照してください。
6. 次のいずれかを行います。
   - 既存のブランチで作業 を選択し、既存のブランチドロップダウンメニューからブランチを選択します。
   - 新しいブランチで作業 を選択し、ブランチ名フィールドにブランチ名を入力し、ドロップダウンメニューから新しいブランチを作成するブランチを選択します。
7. 必要に応じて、開発環境の名前を追加するか、設定を編集します。
8. [作成] を選択します。

CodeCatalyst コンソールでファイルを作成するには

1. ファイルを作成するプロジェクトに移動します。リポジトリに移動する方法の詳細については、「」を参照してくださいソースリポジトリの表示。
2. プロジェクトのソースリポジトリのリストからリポジトリの名前を選択します。または、ナビゲーションペインでコード を選択し、ソースリポジトリ を選択します。

   ファイルを作成するリポジトリを選択します。
3. (オプション) デフォルトのブランチとは異なるブランチにファイルを作成する場合は、ファイルを作成するブランチを選択します。
4. ファイルの作成 を選択します。
5. ファイル名 にファイル名を入力します。エディタに ファイルの内容を追加します。

> **Tip**
>
> ブランチのルートのサブフォルダまたはサブディレクトリにファイルを作成する場合は、ファイル名の一部としてその構造を含めます。

変更に満足したら、コミット を選択します。

6. ファイル名 で、ファイルの名前を確認し、変更を加えます。必要に応じて、ブランチ で使用可能なブランチのリストからファイルを作成するブランチを選択します。コミットメッセージで、この変更を行った理由について、オプションで簡潔で情報の多い説明を入力します。これは、ファイルをソースリポジトリに追加するコミットの基本的なコミット情報として表示されます。
7. コミットを選択してファイルをコミットし、ソースリポジトリにプッシュします。

ローカルコンピュータにクローンを作成し、Git クライアントまたは接続された統合開発環境 (IDE) を使用してファイルや変更をプッシュすることで、ソースリポジトリにファイルを追加することもできます。

> **Note**
>
> Git サブモジュールを追加する場合は、Git クライアントまたは開発環境を使用して git submodule add コマンドを実行する必要があります。 CodeCatalyst コンソールで Git サブモジュールを追加または表示したり、プルリクエストで Git サブモジュールの違いを表示したりすることはできません。Git サブモジュールの詳細については、Git ドキュメント を参照してください。

Git クライアントまたは接続された統合開発環境 (IDE) を使用してファイルを追加するには

1. ソースリポジトリをローカルコンピュータにクローンします。詳細については、「ソースリポジトリのクローン作成」を参照してください。
2. ローカルリポジトリにファイルを作成するか、ローカルリポジトリにファイルをコピーします。
3. 次のいずれかを実行して、コミットを作成してプッシュします。

   - Git クライアントを使用している場合は、ターミナルまたはコマンドラインで コマンドを実行しgit add、追加するファイルの名前を指定します。または、追加または変更されたファイル

をすべて追加するには、 git add コマンドを実行し、その後に 1 つまたは 2 つの期間を実行して、現在のディレクトリレベルですべての変更を含めるか (1 つの期間) 、現在のディレクトリとすべてのサブディレクトリですべての変更を含めるか (2 つの期間) を指定します。変更をコミットするには、 git commit -m コマンドを実行し、コミットメッセージを指定します。の変更を のソースリポジトリにプッシュするには CodeCatalyst、 を実行しますgit push。Git コマンドの詳細については、Git ドキュメントおよび を参照してくださいブランチの Git コマンド。

- 開発環境または IDE を使用している場合は、ファイルを作成して IDE にファイルを追加し、変更をコミットしてプッシュします。詳細については、「」を参照で開発環境を使用してコードを記述および変更する CodeCatalystするか、IDE ドキュメントを参照してください。

ファイルの表示

Amazon CodeCatalyst コンソールでソースリポジトリ内のファイルを表示できます。ファイルは、デフォルトのブランチと他のブランチで表示できます。ファイルの内容は、表示するブランチによって異なる場合があります。

CodeCatalyst コンソールでファイルを表示するには

1. ファイルを表示するプロジェクトに移動します。詳細については、「ソースリポジトリの表示」を参照してください。
2. プロジェクトのソースリポジトリのリストからリポジトリの名前を選択します。または、ナビゲーションペインでコード を選択し、ソースリポジトリ を選択します。

   ファイルを表示するリポジトリを選択します。
3. デフォルトのブランチのファイルとフォルダのリストが表示されます。ファイルは紙のアイコンで示され、フォルダはフォルダのアイコンで示されます。
4. 次のいずれかを実行します。
   - 別のブランチのファイルとフォルダを表示するには、ブランチのリストから選択します。
   - フォルダを展開するには、リストからフォルダを選択します。
5. 特定のファイルの内容を表示するには、リストから選択します。ファイルの内容がブランチに表示されます。別のブランチのファイルの内容を表示するには、ブランチセレクタから目的のブランチを選択します。

> **Tip**
>
> ファイルの内容を表示するときは、ファイルの表示 から表示する追加のファイルを選択できます。ファイルを編集するには、編集 を選択します。

コンソールで複数のファイルを表示できます。Git クライアントまたは統合開発環境 (IDE) を使用して、ローカルコンピュータにクローンしたファイルを表示することもできます。詳細については、Git クライアントまたは IDE のドキュメントを参照してください。

> **Note**
>
> CodeCatalyst コンソールで Git サブモジュールを表示することはできません。Git サブモジュールの詳細については、Git ドキュメント を参照してください。

ファイルの変更履歴の表示

Amazon CodeCatalyst コンソールでソースリポジトリ内のファイルへの変更履歴を表示できます。これにより、ファイルの履歴を表示するブランチに対するさまざまなコミットによってファイルに加えられた変更を理解するのに役立ちます。例えば、ソースリポジトリの **main**ブランチで**readme.md**ファイルへの変更履歴を表示すると、そのブランチでそのファイルへの変更を含むコミットのリストが表示されます。

> **Note**
>
> リンクされたリポジトリ内のファイルの履歴は、 CodeCatalyst コンソールでは表示できません。

CodeCatalyst コンソールでファイルの履歴を表示するには

1. ファイルの履歴を表示するプロジェクトに移動します。詳細については、「ソースリポジトリの表示」を参照してください。
2. プロジェクトのソースリポジトリのリストからリポジトリの名前を選択します。または、ナビゲーションペインでコード を選択し、ソースリポジトリ を選択します。

3. ファイル履歴を表示するリポジトリを選択します。ファイルの履歴を表示するブランチを選択し、リストからファイルを選択します。[View history (履歴の表示)] を選択します。
4. 指定されたブランチで、このファイルへの変更を含むコミットのリストを確認します。特定のコミットに含まれる変更の詳細を表示するには、リストでそのコミットのコミットメッセージを選択します。そのコミットとその親コミットの違いが表示されます。
5. 別のブランチのファイルへの変更履歴を確認するには、ブランチセレクタを使用してそのブランチのビューを変更し、ファイルリストからファイルを選択し、履歴の表示を選択します。

> **Note**
>
> CodeCatalyst コンソールで Git サブモジュールの変更履歴を表示することはできません。Git サブモジュールの詳細については、Git ドキュメント を参照してください。

ファイルの編集

Amazon CodeCatalyst コンソールで個々のファイルを編集できます。複数のファイルを一度に編集するには、開発環境を作成するか、リポジトリのクローンを作成し、Git クライアントまたは統合開発環境 (IDE) を使用して変更を行います。詳細については、で開発環境を使用してコードを記述および変更する CodeCatalystまたはソースリポジトリのクローン作成を参照してください。

CodeCatalyst コンソールでファイルを編集するには

1. ファイルを編集するプロジェクトに移動します。リポジトリに移動する方法の詳細については、「」を参照してくださいソースリポジトリの表示。
2. ファイルを編集するリポジトリを選択します。ブランチを表示 を選択し、作業するブランチを選択します。そのブランチ内のファイルとフォルダのリストからファイルを選択します。

   ファイルの内容が表示されます。
3. [編集] を選択します。
4. エディタで、ファイルの内容を編集し、コミット を選択します。オプションで、コミットメッセージ の変更に関する詳細情報をコミット変更 に追加します。変更に満足したら、コミット を選択します。

ファイルの名前変更または削除

開発環境、コンピュータのローカル、または統合開発環境 (IDE) でファイルの名前を変更または削除できます。ファイルの名前を変更または削除したら、それらの変更をコミットしてソースリポジトリにプッシュします。Amazon CodeCatalyst コンソールでファイルの名前を変更または削除することはできません。

Amazon でのプルリクエストによるコードの確認 CodeCatalyst

プルリクエストは、ユーザーや他のプロジェクトメンバーが、あるブランチから別のブランチへのコード変更を確認、コメント、マージできる主な方法です。プルリクエストを使用して、リリースされたソフトウェアのマイナーな変更や修正、主要な機能の追加、または新しいバージョンについて、コードの変更を共同で確認できます。問題を使用してプロジェクトの作業を追跡する場合は、特定の問題をプルリクエストにリンクして、プルリクエストのコード変更によって対処されている問題を追跡できます。プルリクエストを作成、更新、コメント、マージ、または閉じると、プルリクエストの作成者と、プルリクエストに必要なレビューワーまたはオプションのレビューワーに E メールが自動的に送信されます。

> **Tip**
>
> プロファイルの一部として E メールを受信するプルリクエストイベントを設定できます。詳細については、「Amazon での通知の管理 CodeCatalyst」を参照してください。

プルリクエストには、ソースリポジトリに 2 つのブランチが必要です。1 つはレビューするコードを含むソースブランチで、もう 1 つはレビューしたコードをマージする送信先ブランチです。送信元ブランチには、AFTER コミットが含まれています。これは、送信先ブランチにマージする変更が含まれるコミットです。送信先ブランチには、BEFORE コミットが含まれています。これは、コードの「前」の状態を表しています (プルリクエストブランチが送信先ブランチにマージされる前) 。

> **Note**
>
> プルリクエストの作成中に表示される違いは、ソースブランチのチップと宛先ブランチのチップの違いです。プルリクエストを作成すると、表示される違いは、選択したプルリクエストのリビジョンと、プルリクエストの作成時に送信先ブランチのヒントであったコミットの間になります。Git の違いとマージベースの詳細については、Git ドキュメントgit-merge-baseの「」を参照してください。

特定のソースリポジトリとブランチに対してプルリクエストが作成されている間、プロジェクトの操作の一環としてプルリクエストを作成、表示、確認、および閉じることができます。プルリクエストを表示して操作するために、ソースリポジトリを表示する必要はありません。プルリクエスト状態は、作成時に Open に設定されます。プルリクエストは、 CodeCatalyst コンソールでマージして状態を Merged に変更するか、閉じて状態を Closed に変更するまで開いたままになります。

コードを確認したら、次のいずれかの方法でプルリクエストの状態を変更できます。

- CodeCatalyst コンソールでプルリクエストをマージします。プルリクエストの送信元ブランチのコードは、送信先ブランチにマージされます。プルリクエストのステータスは Merged に変わります。Open に戻すことはできません。
- ブランチをローカルにマージして変更をプッシュし、 CodeCatalyst コンソールでプルリクエストを閉じます。
- CodeCatalyst コンソールを使用して、マージせずにプルリクエストを閉じます。これにより、ステータスが「クローズド」に変更され、ソースブランチから宛先ブランチにコードがマージされることはありません。

プルリクエストを作成する前に、次の操作を実行します。

- レビューするコード変更をコミットしてブランチ (ソースブランチ) にプッシュします。
- プルリクエストの作成時に実行されるワークフローを他のユーザーに通知できるように、プロジェクトの通知を設定します。(このステップはオプションですが、推奨されます)。

トピック

プルリクエストの作成

プルリクエストを作成すると、他のユーザーがコード変更を他のブランチにマージする前にそのコードの変更を確認するのに役立ちます。まず、コード変更のためのブランチを作成します。これは、プルリクエストのソースブランチとして参照されます。変更をコミットしてリポジトリにプッシュした後、ソースブランチの内容を宛先ブランチのコンテンツと比較するプルリクエストを作成できます。

Amazon CodeCatalyst コンソールでは、特定のブランチ、プルリクエストページ、またはプロジェクトの概要からプルリクエストを作成できます。特定のブランチからプルリクエストを作成すると、プルリクエストの作成ページでリポジトリ名とソースブランチが自動的に表示されます。プルリクエストを作成すると、プルリクエストの更新と、プルリクエストがマージまたはクローズされたタイミングに関する E メールが自動的に送信されます。

> **Note**
>
> プルリクエストの作成中に表示される違いは、ソースブランチのチップと宛先ブランチのチップの違いです。プルリクエストが作成されると、表示される違いは、選択したプルリクエストのリビジョンと、プルリクエストの作成時に送信先ブランチのヒントであったコミットの間になります。Git の違いとマージベースの詳細については、Git ドキュメントgit-merge-baseの「」を参照してください。

Amazon Q がプルリクエストに含まれる変更の説明を自動的に作成するようにプルリクエストを作成するときに、「説明の書き込み」機能を使用できます。このオプションを選択すると、Amazon Q はコード変更を含むソースブランチと、これらの変更をマージする送信先ブランチの違いを分析します。次に、それらの変更の概要と、それらの変更の意図と効果の最適な解釈を作成します。この機能は、米国西部 (オレゴン) リージョンでのみ使用できます。

> **Note**
>
> Amazon Bedrock を搭載： AWS 自動不正使用検出を実装。Write description for me と Create content summary 機能は Amazon Bedrock 上に構築されているため、ユーザーは Amazon Bedrock に実装されているコントロールを最大限に活用して、安全性、セキュリティ、人工知能 (AI) の責任ある使用を強制できます。

プルリクエストを作成するには

1. プロジェクトに移動します。

2. 次のいずれかを行います。
   - ナビゲーションペインで、コード を選択し、プルリクエスト を選択し、プルリクエストの作成 を選択します。
   - リポジトリのホームページで、「その他」を選択し、「プルリクエストの作成」を選択します。
   - プロジェクトページで、プルリクエストの作成 を選択します。
3. ソースリポジトリ で、指定されたソースリポジトリがコミットされたコードを含むリポジトリであることを確認します。このオプションは、リポジトリのメインページからプルリクエストを作成していない場合にのみ表示されます。
4. 送信先ブランチ で、レビュー後にコードをマージするブランチを選択します。
5. ソースブランチ で、コミットされたコードを含むブランチを選択します。
6. プルリクエストのタイトル に、他のユーザーがレビューする必要がある内容と理由を理解するのに役立つタイトルを入力します。
7. (オプション) プルリクエストの説明 で、問題へのリンクや変更の説明などの情報を入力します。

> **Tip**
>
> プルリクエストに含まれる変更の説明 CodeCatalyst を自動的に生成するには、説明の書き込み を選択します。プルリクエストに追加した後、自動的に生成された説明を変更できます。
> この機能を使用するには、空間に対して生成 AI 機能を有効にする必要があります。詳細については、「生成 AI 機能の管理」を参照してください。

8. (オプション) 問題 で問題 をリンク を選択し、リストから問題を選択するか、その ID を入力します。問題のリンクを解除するには、リンク解除アイコンを選択します。
9. (オプション) 必要なレビューワー で、必要なレビューワーを追加 を選択します。プロジェクトメンバーのリストから選択して追加します。必要なレビューワーは、プルリクエストを送信先ブランチにマージする前に、変更を承認する必要があります。

> **Note**
>
> レビューワーを必要なレビューワーとオプションのレビューワーの両方として追加することはできません。自分自身をレビューワーとして追加することはできません。

10. (オプション) オプションのレビューワー で、オプションのレビューワーを追加 を選択します。プロジェクトメンバーのリストから選択して追加します。オプションのレビューワーは、プルリクエストを送信先ブランチにマージする前に、変更を要件として承認する必要はありません。
11. ブランチの違いを確認します。プルリクエストに表示される違いは、ソースブランチのリビジョンと、プルリクエストの作成時の送信先ブランチのヘッドコミットであるマージベースとの間の変更です。変更が表示されない場合、ブランチは同一であるか、送信元と送信先の両方に同じブランチを選択している可能性があります。
12. プルリクエストにレビューするコードと変更が含まれていることを確認したら、「 の作成」を選択します。

> **Note**
>
> プルリクエストを作成したら、コメントを追加できます。コメントは、プルリクエスト、またはファイル内の個々の行、およびプルリクエスト全体に追加できます。ファイルなどのリソースへのリンクを追加するには、@ 記号の後にファイルの名前を付けます。

ブランチからプルリクエストを作成するには

1. プルリクエストを作成するプロジェクトに移動します。
2. ナビゲーションペインで、ソースリポジトリ を選択し、コードの変更を確認するブランチを含むリポジトリを選択します。
3. デフォルトのブランチ名の横にあるドロップダウン矢印を選択し、リストから目的のブランチを選択します。リポジトリのすべてのブランチを表示するには、すべての を表示 を選択します。
4. 詳細 を選択し、プルリクエストの作成 を選択します。
5. リポジトリとソースブランチが事前に選択されています。送信先ブランチ で、レビュー後にコードをマージするブランチを選択します。プルリクエストのタイトル に、他のプロジェクトユーザーがレビューする必要がある内容と理由を理解するのに役立つタイトルを入力します。必要に応じて、プルリクエストの説明 に詳細情報を入力します。例えば、 の関連する問題へのリンクに貼り付けたり CodeCatalyst、行った変更の説明を追加したりします。

> **Note**
>
> プルリクエストの作成イベントに対して実行するように設定されたワークフローは、プルリクエストの送信先ブランチがワークフローで指定されたブランチの 1 つと一致する場合、プルリクエストの作成後に実行されます。

6. ブランチの違いを確認します。変更が表示されない場合、ブランチは同じであるか、送信元と送信先の両方に同じブランチを選択している可能性があります。
7. (オプション) 問題 で問題 をリンク を選択し、リストから問題を選択するか、その ID を入力します。問題のリンクを解除するには、リンク解除アイコンを選択します。
8. (オプション) 必要なレビューワー で、必要なレビューワーを追加 を選択します。プロジェクトメンバーのリストから選択して追加します。必要なレビューワーは、プルリクエストを送信先ブランチにマージする前に、変更を承認する必要があります。

> **Note**
>
> レビューワーを必須とオプションの両方として追加することはできません。自分自身をレビューワーとして追加することはできません。

9. (オプション) オプションのレビューワー で、オプションのレビューワーを追加 を選択します。プロジェクトメンバーのリストから選択して追加します。オプションのレビューワーは、プルリクエストを送信先ブランチにマージする前に変更を承認する必要はありません。
10. プルリクエストにレビューする変更が含まれ、必要なレビューワーが含まれていることを確認したら、 の作成 を選択します。

ブランチがプルリクエストの送信先ブランチと一致するように実行するように設定されたワークフローがある場合、プルリクエストの作成後にプルリクエストの詳細エリアの概要にそれらのワークフロー実行に関する情報が表示されます。詳細については、「プッシュ、プル、またはスケジュールトリガーの追加」を参照してください。

プルリクエストの表示

Amazon CodeCatalyst コンソールでプロジェクトのプルリクエストを表示できます。プロジェクト概要ページには、プロジェクトのすべてのオープンプルリクエストが表示されます。状態に関係なくすべてのプルリクエストを表示するには、プロジェクトのプルリクエストページに移動します。プル

リクエストを表示するときに、作成したプルリクエストの変更に関するすべてのコメントの概要を残すことを選択できます。

> **Note**
>
> Amazon Bedrock を搭載： AWS 自動不正使用検出を実装。Write description for me と Create content summary 機能は Amazon Bedrock 上に構築されているため、ユーザーは Amazon Bedrock に実装されているコントロールを最大限に活用して、安全性、セキュリティ、人工知能 (AI) の責任ある使用を強制できます。

オープンプルリクエストを表示するには

1. プルリクエストを表示するプロジェクトに移動します。
2. プロジェクトページには、プルリクエストを作成したユーザー、プルリクエストのブランチを含むリポジトリ、プルリクエストが作成された日付など、オープンプルリクエストが表示されます。オープンプルリクエストビューは、ソースリポジトリでフィルタリングできます。
3. すべてのプルリクエストを表示するには、すべての を表示 を選択します。セレクターを使用して、オプションから選択できます。例えば、すべてのプルリクエストを表示するには、任意のステータス と任意の作成者 を選択します。

   または、ナビゲーションペインでコード を選択し、プルリクエスト を選択し、セレクターを使用してビューを絞り込みます。
4. プルリクエストページで、プルリクエストを ID、タイトル、ステータスなどでソートできます。プルリクエストページに表示される情報と情報の量をカスタマイズするには、歯車アイコンを選択します。
5. 特定のプルリクエストを表示するには、リストから選択します。
6. このプルリクエストに関連付けられたワークフロー実行のステータスを表示するには、概要 を選択し、ワークフロー実行 のプルリクエストの詳細エリアの情報を確認します。

   ワークフローがプルリクエストの作成イベントまたはリビジョンイベントで設定されていて、ワークフローの送信先ブランチの要件がプルリクエストで指定された送信先ブランチと一致する場合、ワークフロー実行が発生します。詳細については、「プッシュ、プル、またはスケジュールトリガーの追加」を参照してください。
7. リンクされた問題がある場合は、概要 を選択し、問題 のプルリクエストの詳細の情報を確認します。リンクされた問題を表示する場合は、リストからその ID を選択します。

8. （オプション) プルリクエストのリビジョンのコード変更に残っているコメントの概要を作成するには、コンテンツ概要の作成 を選択します。概要には、プルリクエスト全体に残っているコメントは含まれません。

> **Note**
>
> この機能を使用するには、生成 AI 機能がスペースに対して有効になっており、米国西部 (オレゴン) リージョンでのみ使用できます。詳細については、「生成 AI 機能の管理」を参照してください。

9. プルリクエストのコード変更を表示するには、変更 を選択します。プルリクエストで変更があったファイルの数と、プルリクエスト内のどのファイルにコメントがあるかは、「ファイルの変更」ですばやく確認できます。フォルダの横に表示されるコメントの数は、そのフォルダ内のファイルに対するコメントの数を示します。フォルダを展開すると、フォルダ内の各ファイルのコメント数が表示されます。特定のコード行に残っているコメントを表示することもできます。

> **Note**
>
> プルリクエストのすべての変更をコンソールに表示することはできません。例えば、コンソールで Git サブモジュールを表示できないため、プルリクエストでサブモジュールの違いを表示することはできません。一部の違いは大きすぎて表示できない場合があります。詳細については、「のソースリポジトリのクォータ CodeCatalyst」および「ファイルの表示」を参照してください。

10. このプルリクエストの品質レポートを表示するには、レポート を選択します。

> **Note**
>
> ワークフローをプルリクエストに表示するには、レポートを生成するように設定する必要があります。詳細については、「ワークフローを使用したテスト」を参照してください。

プルリクエストを承認ルールとマージするための要件の管理

プルリクエストを作成するときに、個々のプルリクエストに必須またはオプションのレビューワーを追加することを選択できます。ただし、特定の送信先ブランチにマージするときに、すべてのプルリ

クエストが満たす必要がある要件を作成することもできます。これらの要件は承認ルールと呼ばれます。承認ルールは、リポジトリ内のブランチに対して設定されます。送信先ブランチに承認ルールが設定されているプルリクエストを作成する場合、プルリクエストをそのブランチにマージする前に、必要なレビューワーの承認に加えて、そのルールの要件が満たされている必要があります。承認ルールを作成すると、デフォルトのブランチなどのブランチへのマージの品質基準を維持するのに役立ちます。

ソースリポジトリのデフォルトブランチに適用される承認ルールは、他のブランチに適用される承認ルールとは少し動作が異なります。デフォルトブランチに適用されるルールは、デフォルトブランチとして指定したブランチに自動的に適用されます。以前にデフォルトブランチとして設定されていたブランチは、そのブランチに適用されたルールを保持します。

承認ルールを作成するときは、現在と将来の両方のプロジェクトユーザーがそのルールをどのように満たすかを考慮する必要があります。例えば、プロジェクトに 6 人のユーザーがいて、送信先ブランチにマージする前に 5 つの承認を必要とする承認ルールを作成すると、プルリケットを作成した人以外の全員がマージする前にそのプルリクエストを承認する必要があるルールを効果的に作成できます。

> **Note**
>
> プロジェクトで承認ルールを作成および管理するには、 CodeCatalyst プロジェクト管理者ロールが必要です。リンクされたリポジトリの承認ルールを作成することはできません。

承認ルールは削除できませんが、ゼロ承認を要求するように更新できるため、ルールは事実上削除されます。

プルリクエストの送信先ブランチの承認ルールを表示および編集するには

1. リポジトリが存在するプロジェクトに移動します。
2. プロジェクトのソースリポジトリのリストからリポジトリの名前を選択します。または、ナビゲーションペインでコード を選択し、ソースリポジトリ を選択します。

   承認ルールを表示するリポジトリを選択します。
3. リポジトリの概要ページで、ブランチ を選択します。
4. 承認ルール 列で、表示 を選択してリポジトリの各ブランチのルールのステータスを表示します。

最小承認数 では、プルリクエストをそのブランチにマージするために必要な承認数に対応します。

5. 承認ルールを作成または変更するには、設定の管理 を選択します。ソースリポジトリの設定ページで、承認ルール で の編集 を選択します。

> **Note**
>
> 承認ルールを編集するには、プロジェクト管理者ロールが必要です。

6. ブランチ で、承認ルールを設定するブランチの名前をドロップダウンリストから選択します。承認の最小数 に数字を入力し、保存 を選択します。

プルリクエストの確認

Amazon CodeCatalyst コンソールを使用して、プルリクエストに含まれる変更を共同で確認してコメントできます。ソースブランチと宛先ブランチの違い、またはプルリクエストのリビジョンの違いで、個々のコード行にコメントを追加できます。他のユーザーが残したフィードバックをすばやく理解できるように、プルリクエストのコード変更に残っているコメントの概要を作成できます。コードを操作する開発環境を作成することもできます。

> **Note**
>
> Amazon Bedrock を搭載： AWS 自動不正使用検出を実装。Write description for me と Create content summary 機能は Amazon Bedrock 上に構築されているため、ユーザーは Amazon Bedrock に実装されているコントロールを最大限に活用して、安全性、セキュリティ、人工知能 (AI) の責任ある使用を強制できます。

> **Tip**
>
> プロファイルの一部として E メールを受信するプルリクエストイベントを設定できます。詳細については、「Amazon での通知の管理 CodeCatalyst」を参照してください。

プルリクエストは、プルリクエストのリビジョンと、プルリクエストの作成時に送信先ブランチのチップであったコミットの違いを示します。これはマージベースと呼ばれます。Git の違いとマージベースの詳細については、Git ドキュメントgit-merge-baseの「」を参照してください。

> **Tip**
>
> コンソールで作業する場合、特にプルリクエストをしばらく開いている場合は、ブラウザを更新して、プルリクエストのレビューを開始する前に最新のリビジョンが使用可能であることを確認することを検討してください。

CodeCatalyst コンソールでプルリクエストを確認するには

1. プロジェクトに移動します。
2. 次のいずれかを実行して、プルリクエストに移動します。
   - プルリクエストがプロジェクトページに一覧表示されている場合は、リストから選択します。
   - プルリクエストがプロジェクトページに表示されない場合は、すべての を表示 を選択します。フィルターとソートを使用してプルリクエストを検索し、リストから選択します。
   - ナビゲーションペインで、コード を選択し、プルリクエスト を選択します。
3. リストからレビューするプルリクエストを選択します。プルリクエストのリストをフィルタリングするには、フィルターバーに名前の一部を入力します。
4. 概要 では、プルリクエストの名前とタイトルを確認できます。プルリクエスト自体に残っているコメントを作成して表示できます。ワークフロー実行、リンクされた問題、レビュー担当者、プルリクエストの作成者、実行可能なマージ戦略に関する情報など、プルリクエストの詳細を表示することもできます。

   > **Note**
   >
   > 特定のコード行に残っているコメントは、変更 に表示されます。

5. (オプション) プルリクエスト全体に適用されるコメントを追加するには、プルリクエストのコメント を展開し、コメントの作成 を選択します。
6. (オプション) このプルリクエストのリビジョンの変更に残っているすべてのコメントの概要を表示するには、コメント概要の作成 を選択します。

> **Note**
>
> この機能を使用するには、生成 AI 機能が空間に対して有効になっており、米国西部 (オレゴン) リージョンでのみ使用できます。詳細については、「生成 AI 機能の管理」を参照してください。

7. 変更 では、送信先ブランチとプルリクエストの最新のリビジョンの違いを確認できます。複数のリビジョンがある場合は、それらの差で比較するリビジョンを変更できます。リビジョンの詳細については、「リビジョン」を参照してください。

> **Tip**
>
> プルリクエストで変更があったファイルの数と、プルリクエスト内のどのファイルにコメントがあるかは、「ファイルの変更」ですばやく確認できます。フォルダの横に表示されるコメントの数は、そのフォルダ内のファイルに対するコメントの数を示します。フォルダを展開すると、フォルダ内の各ファイルのコメント数が表示されます。

8. 相違点の表示方法を変更するには、統合 と分割 のどちらかを選択します。
9. プルリクエストの行にコメントを追加するには、コメントする行に移動します。その行に表示されるコメントアイコンを選択し、コメントを入力し、保存 を選択します。
10. プルリクエスト内のリビジョン間、またはそのソースブランチと宛先ブランチ間の変更を表示するには、「 の比較」で使用可能なオプションから選択します。リビジョンの行に対するコメントは、それらのリビジョンに保持されます。
11. プルリクエストのトリガーでコードカバレッジレポートを生成するようにワークフローを設定している場合は、関連するプルリクエストでラインとブランチカバレッジの結果を表示できます。コードカバレッジの検出結果を非表示にするには、コードカバレッジを非表示 を選択します。詳細については、「コードカバレッジレポート」を参照してください。
12. プルリクエストにコード変更を加える場合は、プルリクエストから開発環境を作成できます。[開発環境を作成] を選択します。オプションで開発環境の名前を追加するか、設定を編集して の作成を選択します。
13. レポート では、このプルリクエストの品質レポートを表示できます。複数のリビジョンがある場合は、それらの差で比較するリビジョンを変更できます。名前、ステータス、ワークフロー、アクション、タイプでレポートをフィルタリングできます。

> **Note**
>
> プルリクエストにレポートが表示されるようにワークフローを設定する必要があります。詳細については、「アクションでの品質レポートの設定」を参照してください。

14. 特定のレポートを表示するには、リストから選択します。詳細については、「ワークフローを使用したテスト」を参照してください。
15. このプルリクエストのレビューワーとしてリストされており、変更を承認する場合は、最新のリビジョンを表示していることを確認し、「承認」を選択します。

> **Note**
>
> 必要なレビューワーはすべて、プルリクエストをマージする前に承認する必要があります。

プルリクエストの更新

プルリクエストを更新することで、他のプロジェクトメンバーがコードを簡単に確認できるようになります。プルリクエストを更新して、レビューワー、問題へのリンク、プルリクエストのタイトル、または説明を変更できます。例えば、プルリクエストに必要なレビューワーを変更して、休暇中の人を削除し、別の人を追加したい場合があります。オープンプルリクエストのソースブランチにコミットをプッシュすることで、コードをさらに変更してプルリクエストを更新することもできます。ソースリポジトリのプルリクエストの CodeCatalyst ソースブランチにプッシュするたびに、リビジョンが作成されます。プロジェクトメンバーは、プルリクエストでリビジョン間の違いを表示できます。

プルリクエストのレビューワーを更新するには

1. プルリクエストのレビューワーを更新するプロジェクトに移動します。
2. プロジェクトページの「プルリクエストを開く」で、レビューワーを更新するプルリクエストを選択します。または、ナビゲーションペインでコード を選択し、プルリクエスト を選択してから、更新するプルリクエストを選択します。
3. (オプション) 概要 のプルリクエストの詳細 エリアで、プラス記号を選択して必須またはオプションのレビューワーを追加します。レビューワーの横にある X を選択して、オプションまたは必要なレビューワーとして削除します。

4. (オプション) 概要 のプルリクエストの詳細 エリアで、問題にリンクを選択してプルリクエストにリンクし、リストから問題を選択するか、その ID を入力します。問題のリンクを解除するには、リンクを解除する問題の横にあるリンク解除アイコンを選択します。

プルリクエストの送信元ブランチのファイルとコードを更新するには

1. 複数のファイルを更新するには、開発環境 を作成するか、リポジトリとそのソースブランチのクローンを作成し、Git クライアントまたは統合開発環境 (IDE) を使用してソースブランチ内のファイルを変更します。変更をコミットしてソースリポジトリの CodeCatalyst ソースブランチにプッシュし、変更でプルリクエストを自動的に更新します。詳細については、「ソースリポジトリのクローン作成」および「Amazon でのコミットによるソースコードの変更について CodeCatalyst」を参照してください。
2. ソースブランチ内の個々のファイルを更新するには、複数のファイルの場合と同様に Git クライアントまたは IDE を使用できます。コンソール CodeCatalystで直接編集することもできます。詳細については、「ファイルの編集」を参照してください。

プルリクエストのタイトルと説明を更新するには

1. プルリクエストのタイトルまたは説明を更新するプロジェクトに移動します。
2. プロジェクトページには、プルリクエストを作成したユーザー、プルリクエストのブランチを含むリポジトリ、プルリクエストが作成された日時など、オープンプルリクエストが表示されます。オープンプルリクエストビューは、ソースリポジトリでフィルタリングできます。リストから変更するプルリクエストを選択します。
3. すべてのプルリクエストを表示するには、すべての を表示 を選択します。または、ナビゲーションペインでコード を選択し、プルリクエスト を選択します。フィルターボックスまたはソート関数を使用して、変更するプルリクエストを検索し、選択します。
4. 概要 で、編集 を選択します。
5. タイトルまたは説明を変更し、保存 を選択します。

プルリクエストのマージ

コードがレビューされ、必要なレビューワーがすべて承認したら、早送りなど、サポートされているマージ戦略を使用して、 CodeCatalyst コンソールでプルリクエストをマージできます。CodeCatalyst コンソールでサポートされているすべてのマージ戦略がすべてのプルリクエストの選択肢として利用できるわけではありません。 CodeCatalyst はマージを評価し、コンソールで利用可

能で、ソースブランチを送信先ブランチにマージできるマージ戦略のみを選択できます。ローカルコンピュータまたは開発環境で git merge コマンドを実行してソースブランチを送信先ブランチにマージすることで、プルリクエストを Git マージ戦略のいずれかとマージすることもできます。その後、送信先ブランチの変更を のソースリポジトリにプッシュできます CodeCatalyst。

> **Note**
>
> ブランチをマージして Git で変更をプッシュしても、プルリクエストは自動的に終了しません。

プロジェクト管理者ロールがある場合は、承認および承認ルールのすべての要件を満たしていないプルリクエストをマージすることもできます。

プルリクエストのマージ (コンソール)

送信元ブランチと送信先ブランチの間にマージの競合がなく、必要なレビューワーがすべてプルリクエストを承認している場合、 CodeCatalyst コンソールでプルリクエストをマージできます。競合がある場合、またはマージを完了できない場合、マージボタンは非アクティブになり、マージ不可ラベルが表示されます。その場合、マージする前に、必要な承認者から承認を取得し、必要に応じて競合をローカルで解決し、それらの変更をプッシュする必要があります。プルリクエストをマージすると、プルリクエストの作成者、および必須またはオプションのレビューワーに E メールが自動的に送信されます。プルリクエストにリンクされた問題のステータスは自動的に終了または変更されません。

> **Tip**
>
> プロファイルの一部として E メールを受信するプルリクエストイベントを設定できます。詳細については、「Amazon での通知の管理 CodeCatalyst」を参照してください。

プルリクエストをマージするには

1. プルリクエストをマージするプロジェクトに移動します。
2. プロジェクトページの「プルリクエストを開く」で、マージするプルリクエストを選択します。プルリクエストが表示されない場合は、すべてのプルリクエストを表示を選択し、リストから選択します。または、ナビゲーションペインでコード を選択し、プルリクエスト を選択してから、マージするプルリクエストを選択します。[Merge (マージ)] を選択します。

3. プルリクエストに使用できるマージ戦略から選択します。オプションで、プルリクエストをマージした後にソースブランチを削除するオプションを選択または選択解除し、マージ を選択します。

> **Note**
>
> マージボタンが非アクティブであるか、マージ不可ラベルが表示されている場合、必要なレビューワーがプルリクエストをまだ承認していないか、 CodeCatalyst コンソールでプルリクエストをマージできません。プルリクエストを承認していないレビューワーは、概要 のプルリクエストの詳細エリアにクロックアイコンで示されます。必要なレビューワーがすべてプルリクエストを承認したが、マージボタンがまだ非アクティブである場合、マージの競合が発生する可能性があります。下線付きの「マージ不可」ラベルを選択すると、プルリクエストをマージできない理由の詳細が表示されます。開発環境または CodeCatalyst コンソールの送信先ブランチのマージ競合を解決してプルリクエストをマージするか、競合を解決してローカルにマージし、マージを含むコミットをの送信元ブランチにプッシュできます CodeCatalyst。詳細については、プルリクエストのマージ (Git)「」および Git ドキュメントを参照してください。

マージ要件を上書きする

プロジェクト管理者ロールがある場合は、必要な承認と承認ルールのすべての要件を満たしていないプルリクエストをマージすることを選択できます。これは、プルリクエストの要件を上書きすることと呼ばれます。必要なレビューワーが利用できない場合や、特定のプルリクエストを、すぐには対応できない承認ルールを持つブランチにマージすることが緊急に必要になった場合に、これを行うことができます。

プルリクエストをマージするには

1. 要件を上書きしてマージするプルリクエストで、マージボタンの横にあるドロップダウン矢印を選択します。「承認要件を上書きする」を選択します。
2. オーバーライドの理由 で、承認ルールと必要なレビューワー要件を満たさずにこのプルリクエストをマージする理由の詳細を入力します。これはオプションですが、強くお勧めします。
3. オプションでマージ戦略を選択するか、デフォルトを受け入れます。自動的に生成されたコミットメッセージを詳細に更新することもできます。

4. マージ時にソースブランチを削除するオプションを選択または選択解除します。プルリクエストをマージする要件を上書きする場合は、他のチームメンバーと決定を確認するまで、ソースブランチを保持することをお勧めします。
5. [Merge (マージ)] を選択します。

プルリクエストのマージ (Git)

Git は、ブランチをマージおよび管理するための多くのオプションをサポートしています。次のコマンドは、使用できるオプションの一部です。詳細については、Git ウェブサイト にある利用可能なドキュメントを参照してください。変更をマージしてプッシュしたら、プルリクエストを手動で閉じます。詳細については、「プルリクエストを閉じる」を参照してください。

ブランチをマージするための一般的な Git コマンド

ローカルリポジトリの送信元ブランチからローカルリポジトリの送信先ブランチに変更をマージします。	`git checkout destination-branch-name`
	
	`git merge source-branch-name`

送信元ブランチを送信先ブランチにマージし、早送りマージを指定します。これにより、ブランチがマージされ、送信先ブランチポインタが送信元ブランチのチップに移動します。	`git checkout destination-branch-name`
	
	`git merge --ff-only source-branch-name`

ソースブランチを送信先ブランチにマージし、スカッシュマージを指定します。これにより、送信元ブランチからのすべてのコミットが、送信先ブランチの 1 つのマージコミットに結合されます。	`git checkout destination-branch-name`
	
	`git merge --squash source-branch-name`

ソースブランチを送信先ブランチにマージし、3 方向マージを指定します。これにより、マージコミットが作成され、送信元ブランチから送信先ブランチに個々のコミットが追加されます。	`git checkout destination-branch-name`
	
	`git merge --no-ff source-branch-name`

ローカルリポジトリのソースブランチを削除します。これは、送信先ブランチにマージし、変	`git branch -d source-branch-name`

更をソースリポジトリにプッシュした後、ローカルリポジトリのクリーンアップとして行うのに役立ちます。	

リモートリポジトリのローカルリポジトリで指定されたニックネームを使用して、リモートリポジトリのソースブランチ (のソースリポジトリ CodeCatalyst) を削除します。(コロン (:) の使用に注意してください)。または、 コマンド`--delete`の一部として を指定します。	`git push remote-name :source-branch-name`
	
	`git push remote-name --delete source-branch-name`

プルリクエストを閉じる

プルリクエストをクローズ済み としてマークできます。これによりプルリクエストはマージされませんが、アクションが必要なプルリクエストと関連性がなくなったプルリクエストを判断するのに役立ちます。これらの変更をマージする予定がなくなった場合、または変更が別のプルリクエストによってマージされた場合は、プルリクエストを閉じることをお勧めします。

プルリクエストを閉じると、プルリクエストの作成者、および必須またはオプションのレビューワーに E メールが自動的に送信されます。プルリクエストにリンクされた問題のステータスは自動的に変更されません。

> **Note**
>
> プルリクエストは、閉じた後で再度開くことはできません。

プルリクエストを閉じるには

1. プルリクエストを閉じるプロジェクトに移動します。
2. プロジェクトページに、オープンプルリクエストが表示されます。閉じるプルリクエストを選択します。
3. [閉じる] を選びます。
4. 情報を確認し、プルリクエストを閉じる を選択します。

Amazon でのコミットによるソースコードの変更について CodeCatalyst

コミットは、内容のスナップショットとリポジトリの内容への変更です。ユーザーが変更をコミットしてブランチにプッシュするたびに、その情報は保存されます。Git コミット情報には、コミット作成者、変更をコミットした人、日時、および行われた変更が含まれます。Amazon CodeCatalyst コンソールでファイルを作成または編集すると、同様の情報が自動的に含まれますが、作成者名は CodeCatalyst ユーザー名です。コミットに Git タグを追加して、特定のコミットを識別することもできます。

Amazon では CodeCatalyst、次のことができます。

- ブランチのコミットのリストを表示します。
- コミットで行われた変更を含む個々のコミットを、その親と比較したときに表示します。

ファイルとフォルダを表示することもできます。詳細については、「Amazon でのソースコードファイルの管理 CodeCatalyst」を参照してください。

トピック

ブランチへのコミットの表示

ブランチに加えられた変更の履歴を表示するには、 CodeCatalyst コンソールでブランチのコミットを確認します。これにより、ブランチに誰がいつ変更したかを把握できます。特定のコミットで行われた変更を確認することもできます。

> **Tip**
>
> 特定のファイルに変更を加えたコミットの履歴を表示することもできます。詳細については、ファイルの表示を参照してください。

Git クライアントを使用してコミットを表示することもできます。詳細については、Git ドキュメントを参照してください。

コミットを表示するには (コンソール)

1. コミットを表示するソースリポジトリを含むプロジェクトに移動します。
2. プロジェクトのソースリポジトリのリストからリポジトリの名前を選択します。または、ナビゲーションペインでコード を選択し、ソースリポジトリ を選択します。

   ブランチへのコミットを表示するリポジトリを選択します。
3. ブランチへの最新のコミットに関する情報を含む、リポジトリのデフォルトブランチが表示されます。コミット を選択します。または、「詳細」を選択し、「コミットの表示」を選択します。
4. 別のブランチのコミットを表示するには、ブランチセレクタを選択し、ブランチの名前を選択します。
5. 特定のコミットの詳細を表示するには、コミットタイトル からそのタイトルを選択します。親コミットに関する情報や、親コミットと指定されたコミットを比較してコードに加えられた変更など、コミットの詳細が表示されます。

> **Tip**
>
> コミットに複数の親がある場合は、親コミット ID の横にあるドロップダウンアイコンを選択して、情報を表示し、変更を表示する親コミットを選択できます。

コミットの表示方法の変更 (CodeCatalyst コンソール)

コミットビューに表示される情報は変更できます。作成者 ID やコミット ID などの列を非表示または表示できます。

コミットの表示方法を変更するには (コンソール)

1. コミットを表示するソースリポジトリを含むプロジェクトに移動します。
2. プロジェクトのソースリポジトリのリストからリポジトリの名前を選択します。または、ナビゲーションペインでコード を選択し、ソースリポジトリ を選択します。

   コミットの表示方法を変更するリポジトリを選択します。
3. ブランチへの最新のコミットに関する情報を含む、リポジトリのデフォルトブランチが表示されます。コミット を選択します。
4. 歯車アイコンを選択します。

5. 設定 で、表示するコミットの数を選択し、コミット作成者、コミット日、コミット ID に関する情報を表示するかどうかを選択します。

> **Note**
>
> 情報の表示でコミットタイトルを非表示にすることはできません。

6. 変更を加えたら、保存 を選択して保存するか、キャンセル を選択して破棄します。

のソースリポジトリのクォータ CodeCatalyst

次の表に、Amazon のソースリポジトリのクォータと制限を示します CodeCatalyst。Amazon のクォータの詳細については、 CodeCatalyst「」を参照してくださいのクォータ CodeCatalyst。

リソース	情報
ブランチ名	使用できる文字の組み合わせの長さは 1～256 文字で、リポジトリ内で一意である必要があります。ブランチ名は以下のようにはできません。 • 先頭および末尾にスラッシュ (/) またはピリオド (.) • @ を 1 文字含める • 2 つ以上の連続するピリオド (..)、スラッシュ (//)、または次の文字の組み合わせ :@{ を含める • スペースまたは以下の文字を含める: ? ^ * [\ ~ : ブランチ名は参照です。ブランチ名の制限の多くは、Git 参照標準に基づいています。詳細については、「Git Internals」および「」を参照してくださいgit-check-ref-format。
プルリクエストに関するコメント	プルリクエストの最大数は 1,000 です。

リソース	情報
コミットメッセージ	最大 1024 文字。

ファイルパス	使用できる文字の組み合わせの長さは 1 ~ 4,096 文字です。ファイルパスは、ファイルおよびそのファイルの正確な場所を特定する間違えのない名前にしてください。ファイルパスは、深さが 20 ディレクトリを超えることはできません。また、ファイルパスでは以下を行うことはできません。
	• 空の文字列を含むこと
	• 相対ファイルパスであること
	• 次の文字のあらゆる組み合わせが含まれていること。
	`/./`
	`/../`
	`//`
	• 末尾にスラッシュまたはバックスラッシュがあること
	
	ファイル名とパスは完全修飾である必要があります。ローカルコンピュータのファイルへの名前とパスは、オペレーティングシステムの基準に従う必要があります。リポジトリ内のファイルへのパスを指定するときは、Amazon Linux の標準を使用します。

ファイルサイズ	コンソールを使用する場合 CodeCatalyst、個々のファイルに対して最大 6 MB。
CodeCatalyst コンソールで表示可能なファイルサイズ	コンソールを使用する場合 CodeCatalyst、個々のファイルに対して最大 6 MB。

リソース	情報
Git の blob サイズ	最大 2 GB です。 **Note** メタデータが 6 MB を超えず、単一の blob が 2 GB を超えない限り、単一コミット内のすべてのファイルの数または合計サイズに制限はありません。ただし、ベストプラクティスとして、1 つの大きなコミットではなく、複数の小さなコミットを作成することを検討してください。
コミットのメタデータ	コミットの合計メタデータ (作成者情報、日付、親コミットリスト、コミットメッセージの組み合わせなど) の最大 6 MB。 **Note** データが 20 MB を超えず、個別ファイルが 6 MB を超えず、単一の blob が 2 GB を超えない限り、単一コミット内のすべてのファイルの数または合計サイズに制限はありません。
プルリクエストにリンクできる CodeCatalyst 問題の数	50
プルリクエストにリンクできる Jira の問題の数	50
スペース内のオープンプルリクエストの数	Amazon CodeCatalyst スペースの最大数は 1,000 です。
スペース内のプルリクエストの合計数	Amazon CodeCatalyst スペースの最大数は 10,000 です。

リソース	情報
一回のプッシュでの参照数	最大 4,000 (作成、削除、および更新を含む)。リポジトリ内の参照の総数に制限はありません。
スペース内のリポジトリの数	Amazon CodeCatalyst スペースの最大数は 5,000 です。
リポジトリの説明	使用できる文字の組み合わせの長さは 0 ~ 1,000 文字です。リポジトリの説明はオプションです。
リポジトリ名	リポジトリ名は、プロジェクト内で一意である必要があります。1 ~ 100 文字の文字、数字、ピリオド、アンダースコア、ダッシュを任意に組み合わせて使用できます。名前では大文字と小文字は区別されません。リポジトリ名は .git で終わることはできません。また、スペースを含めることはできません。また、次の文字を含めることはできません。 ! ? @ # $ % ^ & * ( ) + = { } [ ] | \ / > < ~ ` ' " ; :
リポジトリサイズ	リポジトリのサイズは、スペース全体のストレージ制限の影響を受けます。詳細については、「料金表」と「ソースリポジトリの問題のトラブルシューティング」を参照してください。
プルリクエストのレビューワー	プルリクエストの合計レビューワーは最大 100 人 (オプションまたは必須) 。
プルリクエストの書面による概要	プルリクエストの書き込みサマリーの最大数は、スペースの請求階層によって異なります。詳細については、「の料金」を参照してください。

で開発環境を使用してコードを記述および変更する CodeCatalyst

開発環境はクラウドベースの開発環境です。Amazon では CodeCatalyst、開発環境を使用して、プロジェクトのソースリポジトリに保存されているコードを操作します。開発環境を作成するときは、いくつかのオプションがあります。

- でプロジェクト固有の開発環境を作成し CodeCatalyst 、サポートされている統合開発環境 (IDE) でコードを操作します。
- 空の開発環境を作成し、ソースリポジトリからコードのクローンを作成し、サポートされている IDE でそのコードを操作します。
- IDE で開発環境を作成し、ソースリポジトリを 開発環境にクローンする

devfile は、開発環境を標準化するオープンスタンダード YAML ファイルです。つまり、このファイルは開発環境に必要な開発ツールを体系化します。その結果、開発環境をすばやくセットアップし、プロジェクト間で切り替え、チームメンバー間で開発環境設定をレプリケートできます。開発環境は、特定のプロジェクトのコーディング、テスト、デバッグに必要なすべてのツールを設定する devfile を使用するため、ローカル開発環境の作成と保守に費やす時間を最小限に抑えます。

開発環境に含まれるプロジェクトツールとアプリケーションライブラリは、プロジェクトのソースリポジトリ内の devfile によって定義されます。ソースリポジトリに devfile がない場合、 はデフォルトの devfile CodeCatalyst を自動的に適用します。このデフォルトの devfile には、最も頻繁に使用されるプログラミング言語とフレームワーク用のツールが含まれています。プロジェクトが設計図を使用して作成された場合、 によって devfile が自動的に作成されます CodeCatalyst。devfile の詳細については、https://devfile.io を参照してください。

開発環境を作成したら、自分だけがアクセスできます。開発環境では、サポートされている IDE でソースリポジトリのコードを表示して操作できます。

デフォルトでは、開発環境は 2 コアプロセッサ、4 GB の RAM、16 GB の永続ストレージを持つように設定されています。スペース管理者権限がある場合は、スペースの請求階層を変更して、異なる開発環境設定オプションを使用し、コンピューティングとストレージの制限を管理できます。

トピック

- 開発環境の再開
- 開発環境の編集
- 開発環境の削除
- SSH を使用した開発環境への接続
- 開発環境用の devfile の設定
- VPC 接続を開発環境に関連付ける
- の開発環境のクォータ CodeCatalyst

開発環境の作成

開発環境は複数の方法で作成できます。

- 概要ページ、開発環境ページ、またはソースリポジトリページから、 CodeCatalyst ソースリポジトリまたはリンクされたソースリポジトリ CodeCatalyst を使用して で開発環境を作成する
- 開発環境ページからソースリポジトリに接続 CodeCatalyst されていない空の開発環境を に作成する
- 選択した IDE に開発環境を作成し、任意のソースリポジトリを開発環境にクローンする

リポジトリのブランチごとに 1 つの開発環境を作成できます。プロジェクトは複数のリポジトリを持つことができます。作成する開発環境は CodeCatalyst アカウントでのみ管理できますが、開発環境を開いて、サポートされている任意の IDEs で操作できます。IDE で開発環境を使用するには、AWS Toolkit がインストールされている必要があります。詳細については、「開発環境でサポートされる統合開発環境」を参照してください。デフォルトでは、開発環境は 2 コアプロセッサ、4 GB の RAM、16 GB の永続ストレージで作成されます。

> **Note**
>
> ソースリポジトリに関連付けられている開発環境を作成した場合、リソース列には、この開発環境の作成時に指定したブランチが常に表示されます。これは、別のブランチを作成する場合、開発環境内の別のブランチに切り替える場合、または追加のリポジトリのクローンを作成する場合にも適用されます。空の開発環境を作成した場合、リソース列は空白になります。

開発環境でサポートされる統合開発環境

開発環境は、以下のサポートされている統合開発環境 (IDEsで使用できます。

- AWS Cloud9
- JetBrains IDEs
  - IntelliJ IDEA Ultimate
  - GoLand
  - PyCharm プロフェッショナル
- Visual Studio Code

での開発環境の作成 CodeCatalyst

で開発環境の使用を開始するには CodeCatalyst、 AWS Builder ID または SSO を使用して認証し、サインインします。

ブランチから開発環境を作成するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 開発環境を作成するプロジェクトに移動します。
3. ナビゲーションペインで、次のいずれかを実行します。
   - 概要 を選択し、「開発環境」セクションに移動します。
   - コード を選択し、開発環境 を選択します。
   - コード を選択し、ソースリポジトリ を選択し、開発環境を作成するリポジトリを選択します。
4. [開発環境を作成] を選択します。
5. ドロップダウンメニューからサポートされている IDE を選択します。詳細については、「開発環境でサポートされる統合開発環境」を参照してください。
6. [リポジトリのクローン] を選択します。
7. 次のいずれかを行います。
   a. クローンするリポジトリを選択し、[既存のブランチで作業する] を選択し、[既存のブランチ] ドロップダウンメニューからブランチを選択します。

> **Note**
>
> サードパーティーのリポジトリを選択する場合は、既存のブランチで作業する必要があります。

b. クローンするリポジトリを選択し、[新しいブランチで作業する] を選択し、[ブランチ名] フィールドにブランチ名を入力し、[ブランチの作成元] ドロップダウンメニューから新しいブランチを作成するブランチを選択します。

> **Note**
>
> ソースリポジトリページまたは特定のソースリポジトリから開発環境を作成する場合、リポジトリを選択する必要はありません。開発環境は、ソースリポジトリページから選択したソースリポジトリから作成されます。

8. (オプション) エイリアス - オプションで、開発環境のエイリアスを入力します。
9. (オプション) 開発環境設定の編集ボタンを選択して、開発環境のコンピューティング、ストレージ、またはタイムアウト設定を編集します。
10. (オプション) Amazon Virtual Private Cloud (Amazon VPC) - オプション で、ドロップダウンメニューから開発環境に関連付ける VPC 接続を選択します。

    スペースにデフォルト VPC が設定されている場合、開発環境はその VPC に接続された状態で実行されます。別の VPC 接続を関連付けることで、これを上書きできます。また、VPC に接続された開発環境は をサポートしていないことに注意してください AWS Toolkit。

> **Note**
>
> VPC 接続を使用して開発環境を作成すると、関連付けられた VPC ロールを使用して、VPC. CodeCatalyst interacts とこのインターフェイス内に新しいネットワークインターフェイスが作成されます。また、IPv4 CIDR ブロックが `172.16.0.0/12` IP アドレス範囲に設定されていないことを確認します。

11. [作成] を選択します。開発環境の作成中は、開発環境のステータス列に [開始中] と表示され、開発環境が作成されると、ステータス列に [実行中] と表示されます。

空の開発環境を作成するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 開発環境を作成するプロジェクトに移動します。
3. ナビゲーションペインで、次のいずれかを実行します。
   - 概要 を選択し、「開発環境」セクションに移動します。
   - コード を選択し、開発環境 を選択します。
4. [開発環境を作成] を選択します。
5. ドロップダウンメニューからサポートされている IDE を選択します。詳細については、「開発環境でサポートされる統合開発環境」を参照してください。
6. 空の開発環境の作成 を選択します。
7. (オプション) エイリアス - オプションで、開発環境のエイリアスを入力します。
8. (オプション) 開発環境設定の編集ボタンを選択して、開発環境のコンピューティング、ストレージ、またはタイムアウト設定を編集します。
9. (オプション) Amazon Virtual Private Cloud (Amazon VPC) - オプションで、ドロップダウンメニューから開発環境に関連付ける VPC 接続を選択します。

   スペースにデフォルト VPC が設定されている場合、開発環境はその VPC に接続された状態で実行されます。別の VPC 接続を関連付けることで、これを上書きできます。また、VPC に接続された開発環境は をサポートしていないことに注意してください AWS Toolkit。

   > **Note**
   >
   > VPC 接続を使用して開発環境を作成すると、関連付けられた VPC ロールを使用して、VPC. CodeCatalyst interacts とこのインターフェイス内に新しいネットワークインターフェイスが作成されます。また、IPv4 CIDR ブロックが `172.16.0.0/12` IP アドレス範囲に設定されていないことを確認します。

10. [作成] を選択します。開発環境の作成中は、開発環境のステータス列に [開始中] と表示され、開発環境が作成されると、ステータス列に [実行中] と表示されます。

> **Note**
>
> 開発環境を初めて作成して開くには、1～2 分かかる場合があります。

> **Note**
>
> 開発環境が IDE で開いたら、コードに変更をコミットしてプッシュする前に、ディレクトリをソースリポジトリに変更する必要がある場合があります。

IDE で開発環境を作成する

開発環境を使用すると、プロジェクトのソースリポジトリに保存されているコードをすばやく操作できます。開発環境では、サポートされている統合開発環境 (IDE) を使用して、プロジェクト固有の完全に機能するクラウド開発環境ですぐにコーディングを開始できるため、開発速度が向上します。

IDE CodeCatalyst から を操作する方法については、次のドキュメントを参照してください。

- Amazon CodeCatalyst JetBrains IDEs
- Amazon CodeCatalyst for VS Code
- Amazon CodeCatalyst for AWS Cloud9

開発環境の停止

開発環境の`/projects`ディレクトリには、ソースリポジトリからプルされたファイルと、開発環境の設定に使用される devfile が保存されます。開発環境の作成時に空になる `/home` ディレクトリには、開発環境の使用中に作成したファイルが保存されます。開発環境の `/projects`および `/home` ディレクトリのすべてのものは永続的に保存されるため、別の開発環境、リポジトリ、またはプロジェクトに切り替える必要がある場合は、開発環境での作業を停止できます。

> **Warning**
>
> ウェブブラウザ、リモートシェル、IDEs などのインスタンスが接続されたままになっても、開発環境はタイムアウトしません。そのため、追加コストが発生しないように、接続されているすべてのインスタンスを必ず閉じてください。

開発環境の作成中にタイムアウトフィールドで選択された時間アイドル状態になると、開発環境は自動的に停止します。開発環境はアイドル状態になる前に停止できます。開発環境の作成時にタイムアウトなしを選択した場合、開発環境は自動的に停止しません。代わりに、継続的に実行されます。

> **Warning**
>
> 削除された VPC 接続に関連付けられている開発環境を停止した場合、再開することはできません。

開発環境ページから開発環境を停止するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 開発環境を停止するプロジェクトに移動します。
3. ナビゲーションペインで、コード を選択します。
4. 開発環境 を選択します。
5. 停止する開発環境のラジオボタンを選択します。
6. アクションメニューから、停止 を選択します。

> **Note**
>
> コンピューティングの使用は開発環境の実行中にのみ課金されますが、ストレージの使用は開発環境が存在する期間全体にわたって課金されます。開発環境がコンピューティング請求を停止するために使用されていない場合は、開発環境を停止します。

開発環境の再開

開発環境の`/projects`ディレクトリには、ソースリポジトリからプルされたファイルと、開発環境の設定に使用される devfile が保存されます。開発環境の作成時に空になる /home ディレクトリには、開発環境の使用中に作成したファイルが保存されます。開発環境の `/projects`および /home ディレクトリのすべてのものは永続的に保存されるため、別の開発環境、リポジトリ、またはプロジェクトに切り替えて、後で開発環境で作業を再開する必要がある場合は、開発環境での作業を停止できます。

開発環境の作成中にタイムアウトフィールドで選択された時間アイドル状態になると、開発環境は自動的に停止します。開発環境をアイドル状態にするには、 AWS Cloud9 ブラウザタブを閉じる必要があります。

> **Note**
>
> 開発環境を作成したブランチを削除しても、開発環境は引き続き利用可能で実行されています。ブランチを削除した開発環境で作業を再開する場合は、新しいブランチを作成し、そのブランチに変更をプッシュします。

概要ページから開発環境を再開するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 開発環境を再開するプロジェクトに移動し、「開発環境」セクションに移動します。
3. (IDE) で再開を選択します。
   - JetBrains IDEs、アプリケーションを選択して JetBrains ゲートウェイリンク を開くように求められたら、 JetBrainsゲートウェイ EAP を選択します。「リンクを開く」を選択して、プロンプトが表示されたら確認します。
   - VS Code IDE では、アプリケーションを選択して VS Code リンクを開くように求められたら、VS Code を選択します。Open Link を選択して確定します。

ソースリポジトリから開発環境を再開するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 開発環境を再開するプロジェクトに移動します。
3. ナビゲーションペインで、コード を選択します。
4. ソースリポジトリ を選択します。
5. 再開する開発環境を含むソースリポジトリを選択します。
6. ブランチ名を選択してブランチのドロップダウンメニューを表示し、ブランチを選択します。
7. 開発環境を再開 を選択します。
   - JetBrains IDEs の場合、リンクを開くを選択して、このサイトにゲートウェイとの JetBrains JetBrainsゲートウェイリンクを開くことを許可するように求められたら確認します。
   - VS Code IDE では、リンクを開くを選択して、このサイトに Visual Studio Code との VS Code リンクを開くことを許可するように求められたら確認します。

開発環境ページから開発環境を再開するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 開発環境を再開するプロジェクトに移動します。
3. ナビゲーションペインで、コード を選択します。
4. 開発環境 を選択します。
5. IDE 列から、開発環境の (IDE) で再開を選択します。
   - JetBrains IDEs の場合、リンクを開くを選択して、このサイトにゲートウェイとの JetBrains JetBrainsゲートウェイリンクを開くことを許可するように求められたら確認します。
   - VS Code IDE では、リンクを開くを選択して、このサイトに Visual Studio Code との VS Code リンクを開くことを許可するように求められたら確認します。

> **Note**
>
> 開発環境の再開には数分かかることがあります。

開発環境の編集

IDE の実行中に開発環境を編集できます。コンピューティングタイムアウトまたは非アクティブタイムアウトを編集すると、変更を保存した後に開発環境が再起動します。

開発環境を編集するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 開発環境を編集するプロジェクトに移動します。
3. ナビゲーションペインで、コード を選択します。
4. 開発環境 を選択します。
5. 編集する開発環境を選択します。
6. [編集] を選択します。
7. コンピューティングまたは非アクティブタイムアウトに必要な変更を加えます。
8. [保存] を選択します。

開発環境の削除

開発環境に保存されているコンテンツの作業が完了したら、開発環境を削除できます。新しい開発環境を作成して、新しいコンテンツを処理します。開発環境を削除すると、永続コンテンツは完全に削除されます。開発環境を削除する前に、コードの変更をコミットして開発環境の元のソースリポジトリにプッシュしてください。開発環境を削除すると、開発環境のコンピューティングとストレージの請求は停止します。

開発環境を削除した後、ストレージクォータが更新されるまでに数分かかる場合があります。ストレージクォータに達した場合、この間、新しい開発環境を作成することはできません。

> **Important**
>
> 開発環境の削除は元に戻すことができません。開発環境を削除すると、復元できなくなります。

開発環境を削除するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 開発環境を削除するプロジェクトに移動します。
3. ナビゲーションペインで、コード を選択します。
4. 開発環境 を選択します。
5. 削除する開発環境を選択します。
6. [削除] を選択します。
7. **`delete`** を入力して開発環境の削除を確認します。
8. [削除] を選択します。

> **Note**
>
> スペース内の VPC 接続を削除する前に、その VPC に関連付けられている開発環境を削除してください。
> 開発環境を削除しても、VPC 内のネットワークインターフェイスを削除しない場合があります。必要に応じてリソースをクリーンアップしてください。VPC に接続された開発環境を削

除したときにエラーが発生した場合は、古い接続をデタッチし、使用されていないことを確認した後に削除する必要があります。

SSH を使用した開発環境への接続

SSH を使用して開発環境に接続し、ポート転送、ファイルのアップロードとダウンロード、その他の IDEs の使用など、制限なくアクションを実行できます。

> **Note**
>
> IDE タブまたはウィンドウを閉じた後に SSH を長時間使用し続ける場合は、IDE で非アクティブが原因で停止しないように、開発環境のタイムアウトを長く設定してください。

前提条件

- 次のいずれかのオペレーティングシステムが必要です。
  - Windows 10 以降で OpenSSH が有効になっている
  - macOS および Bash バージョン 3 以降
  - yum、、dpkgまたはrpmパッケージマネージャーと Bash バージョン 3 以降の Linux
- AWS CLI バージョン 2.9.4 以降も必要です。

SSH を使用して開発環境に接続するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. SSH を使用して開発環境に接続するプロジェクトに移動します。
3. ナビゲーションペインで、コード を選択します。
4. 開発環境 を選択します。
5. SSH を使用して接続する実行中の開発環境を選択します。
6. SSH 経由で接続 を選択し、目的のオペレーティングシステムを選択し、以下を実行します。
   - まだ行っていない場合は、指定したターミナルで最初のコマンドを貼り付けて実行します。コマンドはスクリプトをダウンロードし、ローカル環境で次の変更を実行して、SSH を使用して開発環境に接続できるようにします。
     - の Session Manager プラグインをインストールします AWS CLI。

  - SSO ログインを実行できるように、ローカル を変更 AWS Config し、 CodeCatalyst プロファイルを追加します。詳細については、「AWS CLIとを使用するためのセットアップ CodeCatalyst」を参照してください。
  - ローカル SSH 設定を変更し、SSH を使用して開発環境に接続するために必要な設定を追加します。
  - SSH クライアントが開発環境に接続するために使用するスクリプトを `~/.aws/codecatalyst-dev-env` ディレクトリに追加します。このスクリプトは CodeCatalyst StartDevEnvironmentSession API を呼び出し、 AWS Systems Manager Session Manager プラグインを使用して開発環境とのセッションを確立します。この AWS Systems Manager セッションは、ローカル SSH クライアントがリモート開発環境に安全に接続するために使用されます。
- 2 番目のコマンド CodeCatalyst を使用して AWS SSO を使用して Amazon にサインインします。このコマンドは、 `~/.aws/codecatalyst-dev-env` ディレクトリのスクリプトが CodeCatalyst StartDevEnvironmentSession API を呼び出すことができるように、認証情報をリクエストして取得します。このコマンドは、認証情報の有効期限が切れるたびに実行する必要があります。モーダル (ssh <destination>) で最後のコマンドを実行すると、認証情報の有効期限が切れているか、このステップで指示されているように SSO ログインを実行していない場合にエラーが発生します。
- 3 番目のコマンドを使用して SSH を使用して、指定した開発環境に接続します。このコマンドの構造は次のとおりです。

```
ssh codecatalyst-dev-env=<space-name>=<project-name>=<dev-environment-id>
```

  また、このコマンドを使用して、ポート転送やファイルのアップロードやダウンロードなど、SSH クライアントで許可されるその他のアクションを実行することもできます。

  - ポート転送：

```
ssh -L <local-port>:127.0.0.1:<remote-port> codecatalyst-dev-env=<space-name>=<project-name>=<dev-environment-id>
```

  - 開発環境のホームディレクトリにファイルをアップロードする：

```
scp -0 </path-to-local-file> codecatalyst-dev-env=<space-name>=<project-name>=<dev-environment-id>:</path-to-remote-file-or-directory>
```

開発環境用の devfile の設定

devfile は、チーム全体の開発環境をカスタマイズするのに役立つオープンスタンダードです。devfile は、必要な開発ツールをコーディングする YAML ファイルです。devfile を設定することで、必要なプロジェクトツールとアプリケーションライブラリを事前に決定でき、Amazon はそれらを開発環境に CodeCatalyst インストールします。devfile は、作成されたリポジトリに固有であり、リポジトリごとに個別の devfile を作成できます。開発環境はコマンドとイベントをサポートし、デフォルトのユニバーサル devfile イメージを提供します。

空のブループリントを使用してプロジェクトを作成する場合は、devfile を手動で作成できます。別のブループリントを使用してプロジェクトを作成すると、 によって devfile が自動的に CodeCatalyst 作成されます。開発環境の`/projects`ディレクトリには、ソースリポジトリと devfile からプルされたファイルが保存されます。開発環境を初めて作成するときに空になる /home ディレクトリには、開発環境の使用中に作成したファイルが保存されます。開発環境の `/projects`および /home ディレクトリのすべてのものは永続的に保存されます。

> **Note**
>
> /home フォルダは、devfile または devfile コンポーネント名の名前を変更する場合にのみ変更されます。devfile または devfile コンポーネント名を変更すると、/homeディレクトリの内容が置き換えられ、以前の/homeディレクトリデータは復元できません。

ルートに devfile を含まないソースリポジトリを使用して開発環境を作成する場合、またはソースリポジトリなしで開発環境を作成する場合、デフォルトのユニバーサル devfile がソースリポジトリに自動的に適用されます。すべての IDEs。 CodeCatalyst 現在、 は devfile バージョン 2.0.0 をサポートしています。devfile の詳細については、「Devfile スキーマ - バージョン 2.0.0」を参照してください。

> **Note**
>
> devfile にはパブリックコンテナイメージのみを含めることができます。

VPC に接続された開発環境では、次の devfile イメージのみがサポートされることに注意してください。

- ユニバーサルイメージ

- リポジトリが VPC と同じリージョンにある場合のプライベート Amazon ECR イメージ

トピック

開発環境のリポジトリ devfile の編集

次の手順を使用して、開発環境のリポジトリ devfile を編集します。

で開発環境のリポジトリ devfile を編集する CodeCatalyst

リポジトリ devfile を編集するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. devfile を編集するソースリポジトリを含むプロジェクトに移動します。
3. ナビゲーションペインで、コード を選択します。
4. ソースリポジトリ を選択します。
5. 編集する devfile を含むソースリポジトリを選択します。
6. ファイルのリストからファイルを選択しますdevfile.yaml。
7. [編集] を選択します。
8. devfile を編集します。
9. コミット を選択するか、プルリクエストを作成して、チームメンバーが変更を確認して承認できるようにします。

> **Note**
>
> devfile を編集する場合は、変更を有効にするために devfile を再起動する必要があります。これを行うには、 を実行します`/aws/mde/mde start --location devfile.yaml`。devfile の開始に問題がある場合、復旧モードになります。ただし、VPC に接続された開発環境に関連付けられた devfile を編集する場合、変更を有効にするには、代わりに開発環境を再起動する必要があります。

を実行して、どの devfile が使用されているかを確認できます`/aws/mde/mde status`。場所フィールドには、環境の`/projects`フォルダに対する devfile のパスがあります。

```
{
            "status": "STABLE",
            "location": "devfile.yaml"
        }
```

デフォルトの devfile を`/projects/devfile.yaml`ソースコードリポジトリに移動することもできます。devfile の場所を更新するには、 コマンド を使用します`/aws/mde/mde start --location` *`repository-name`*`/devfile.yaml`。

IDE で開発環境のリポジトリ devfile を編集する

開発環境の設定を変更するには、devfile を編集する必要があります。サポートされている IDE で devfile を編集してから開発環境を更新することをお勧めしますが、 のソースリポジトリのルートから devfile を編集することもできます CodeCatalyst。サポートされている IDE で devfile を編集する場合は、変更をコミットしてソースリポジトリにプッシュするか、プルリクエストを作成して、チームメンバーが devfile の編集を確認して承認できるようにする必要があります。

- で開発環境のリポジトリ devfile を編集する AWS Cloud9
- VS Code で開発環境のリポジトリ devfile を編集する
- で開発環境のリポジトリ devfile を編集する JetBrains

でサポートされている開発ファイル機能 CodeCatalyst

CodeCatalyst は、バージョン 2.0.0 で次の devfile 機能をサポートしています。devfile の詳細については、「Devfile スキーマ - バージョン 2.0.0」を参照してください。

機能	タイプ
exec	Command
postStart	イベント
container	コンポーネント
args	コンポーネントのプロパティ
env	コンポーネントのプロパティ
mountSources	コンポーネントのプロパティ
volumeMounts	コンポーネントのプロパティ

開発環境の devfile の例

以下は、シンプルな devfile の例です。

```
schemaVersion: 2.0.0
metadata:
  name: al2
components:
  - name: test
    container:
      image: public.ecr.aws/amazonlinux/amazonlinux:2
      mountSources: true
      command: ['sleep', 'infinity']
  - name: dockerstore
commands:
  - id: setupscript
    exec:
      component: test
      commandLine: "chmod +x script.sh"
      workingDir: /projects/devfiles
  - id: executescript
    exec:
      component: test
      commandLine: "/projects/devfiles/script.sh"
  - id: yumupdate
```

```
    exec:
      component: test
      commandLine: "yum -y update --security"
events:
  postStart:
    - setupscript
    - executescript
    - yumupdate
```

Devfile の起動、コマンド、およびイベントログはキャプチャされ、 に保存されます/aws/mde/logs。devfile の動作をデバッグするには、動作中の devfile を使用して開発環境を起動し、ログにアクセスします。

リカバリモードを使用したリポジトリ devfile のトラブルシューティング

devfile の起動に問題がある場合は、復旧モードになり、環境に接続して devfile を修正できます。復旧モードの間、 を実行する/aws/mde/mde statusと devfile の場所は含まれません。

```
{
            "status": "STABLE"
        }
```

のログでエラーを確認し/aws/mde/logs、devfile を修正して、/aws/mde/mde startもう一度実行してみてください。

開発環境用のユニバーサル devfile イメージの指定

デフォルトのユニバーサルイメージには、IDE に使用できる最も一般的に使用されるプログラミング言語と関連ツールが含まれています。イメージが指定されていない場合、 はこのイメージ CodeCatalyst を提供し、 によって管理されるツールを含みます CodeCatalyst。新しいイメージリリースの通知を受けるには、「」を参照してくださいSNS を使用したユニバーサルイメージ通知のサブスクライブ。

> **Note**
>
> public.ecr.aws/aws-mde/universal-image:latest イメージはpublic.ecr.aws/aws-mde/universal-image:3.0イメージを取得します。

Amazon では、以下の devfile イメージ CodeCatalyst がサポートされています。

イメージバージョン	イメージ識別子
Universal image 1.0	public.ecr.aws/aws-mde/universal-image:1.0
Universal image 2.0	public.ecr.aws/aws-mde/universal-image:2.0
Universal image 3.0	public.ecr.aws/aws-mde/universal-image:3.0

> **Note**
>
> を使用している場合 AWS Cloud9、 にアップグレードした後、PHP、Ruby、および CSS ではオートコンプリートは機能しませんuniversal-image:3.0。

トピック

- SNS を使用したユニバーサルイメージ通知のサブスクライブ
- ユニバーサルイメージ 1.0 ランタイムバージョン
- ユニバーサルイメージ 2.0 ランタイムバージョン
- ユニバーサルイメージ 3.0 ランタイムバージョン

SNS を使用したユニバーサルイメージ通知のサブスクライブ

CodeCatalyst はユニバーサルイメージ通知サービスを提供します。これを使用して、 CodeCatalyst ユニバーサルイメージ更新がリリースされたときに通知する Amazon Simple Notification Service (SNS) トピックをサブスクライブできます。SNS トピックの詳細については、「Amazon Simple Notification Service とは」を参照してください。

新しいユニバーサルイメージがリリースされるたびに、サブスクライバーに通知が送信されます。このセクションでは、 CodeCatalyst ユニバーサルイメージの更新をサブスクライブする方法について説明します。

サンプルメッセージ

```
{
    "Type": "Notification",
    "MessageId": "123456789",
    "TopicArn": "arn:aws:sns:us-east-1:1234657890:universal-image-updates",
    "Subject": "New Universal Image Release",
    "Message": {
        "v1": {
            "Message": "A new version of the Universal Image has been released. You are
 now able to launch new DevEnvironments using this image.",
            "image ": {
                "release_type": "MAJOR VERSION",
                "image_name": "universal-image",
                "image_version": "2.0",
                "image_uri": "public.ecr.aws/amazonlinux/universal-image:2.0"
            }
        }
    },
    "Timestamp": "2021-09-03T19:05:57.882Z",
    "UnsubscribeURL": "example url"
}
```

Amazon SNS コンソールを使用して CodeCatalyst ユニバーサルイメージ更新をサブスクライブするには

1. Amazon SNS コンソールを開き、ダッシュボード を開きます。
2. ナビゲーションバーで、 を選択します AWS リージョン。
3. ナビゲーションペインで、[Subscriptions] (サブスクリプション) を選択して、[Create subscription] (サブスクリプションの作成) を選択します。
4. トピック ARN で、 と入力しますarn:aws:sns:us-east-1:089793673375:universal-image-updates。
5. [プロトコル] で、[E メール] を選択します。
6. エンドポイント で、E メールアドレスを指定します。この E メールアドレスは、通知の受信に使用されます。
7. [サブスクリプションを作成] を選択します。
8. AWS 「通知 - サブスクリプションの確認」という件名の確認メールが届きます。E メールを開き、サブスクリプションの確認 を選択します。

Amazon SNS コンソールを使用して CodeCatalyst ユニバーサルイメージ更新のサブスクリプションを解除するには

1. Amazon SNS コンソールを開き、ダッシュボード を開きます。
2. ナビゲーションバーで、 を選択します AWS リージョン。
3. ナビゲーションペインで、サブスクリプションを選択し、サブスクリプションを解除するサブスクリプションを選択します。
4. アクション を選択し、サブスクリプションの削除 を選択します。
5. [削除] を選択します。

ユニバーサルイメージ 1.0 ランタイムバージョン

次の表に、 で使用可能なランタイムを示しますuniversal-image:1.0。

universal-image:1.0 ランタイムバージョン

ランタイム名	Version	特定のメジャーバージョン と最新のマイナーバージョン
AWS CLI	2.11	`aws-cli: 2.x`
Docker 構成	2.16	`docker-compose: 2.x`
dotnet	6.0	`dotnet: 6.x`
	7.0	`dotnet: 7.x`
golang	1.19	`golang: 1.x`
java	corretto11	`java: corretto11.x`
	corretto17	`java: corretto17.x`
nodejs	14.20	`nodejs: 14.x`
	16.19	`nodejs: 16.x`
openssl	1.0	`openssl: 1.x`
	1.1	

ランタイム名	Version	特定のメジャーバージョン と最新のマイナーバージョン
php	7.2	`php: 7.x`
python	3.9	`python: 3.x`
	3.10	
ruby	3.1	`ruby: 3.x`
terraform	1.4	`terraform: 1.x`

ユニバーサルイメージ 2.0 ランタイムバージョン

次の表に、 で使用可能なランタイムを示しますuniversal-image:2.0。

universal-image:2.0 ランタイムバージョン

ランタイム名	Version	特定のメジャーバージョン と最新のマイナーバージョン
AWS CLI	2.11	`aws-cli: 2.x`
Docker 構成	2.17	`docker-compose: 2.x`
dotnet	6.0	`dotnet: 6.x`
	7.0	`dotnet: 7.x`
golang	1.20	`golang: 1.x`
java	corretto11	`java: corretto11.x`
	corretto17	`java: corretto17.x`
nodejs	16.19	`nodejs: 16.x`
openssl	1.0	`openssl: 1.x`
	1.1	

ランタイム名	Version	特定のメジャーバージョン と最新のマイナーバージョン
php	7.2	`php: 7.x`
python	3.9	`python: 3.x`
	3.10	
ruby	3.2	`ruby: 3.x`
terraform	1.4	`terraform: 1.x`

ユニバーサルイメージ 3.0 ランタイムバージョン

次の表に、 で使用可能なランタイムを示しますuniversal-image:3.0。

universal-image:3.0 ランタイムバージョン

ランタイム名	Version	特定のメジャーバージョン と最新のマイナーバージョン
AWS CLI	2.11	`aws-cli: 2.x`
Docker 構成	2.17	`docker-compose: 2.x`
dotnet	6.0	`dotnet: 6.x`
	7.0	`dotnet: 7.x`
golang	1.21	`golang: 1.x`
java	corretto11	`java: corretto11.x`
	corretto17	`java: corretto17.x`
nodejs	18.17	`nodejs: 18.x`
	20.6	`nodejs: 20.x`
openssl	3.0	`openssl: 3.x`

ランタイム名	Version	特定のメジャーバージョン と最新のマイナーバージョン
php	8.2	`php: 8.x`
python	3.9	`python: 3.x`
	3.11	
ruby	3.2	`ruby: 3.x`
terraform	1.5	`terraform: 1.x`

Devfile コマンド

現在、 は devfile 内の`exec`コマンド CodeCatalyst のみをサポートしています。詳細については、Devfile.io ドキュメントの「コマンドの追加」を参照してください。

次の例は、devfile で`exec`コマンドを指定する方法を示しています。

```
commands:
  - id: setupscript
    exec:
      component: test
      commandLine: "chmod +x script.sh"
      workingDir: /projects/devfiles
  - id: executescript
    exec:
      component: test
      commandLine: "./projects/devfiles/script.sh"
  - id: updateyum
    exec:
      component: test
      commandLine: "yum -y update --security"
```

開発環境に接続したら、ターミナルから定義済みのコマンドを実行できます。

```
/aws/mde/mde command <command-id>
/aws/mde/mde command executescript
```

長時間実行されるコマンドの場合、ストリーミングフラグを使用してコマンドの実行をリアルタイムで-s出力できます。

```
/aws/mde/mde -s command <command-id>
```

> **Note**
>
> command-id は小文字にする必要があります。

でサポートされている Exec パラメータ CodeCatalyst

CodeCatalyst は、devfile バージョン 2.0.0 で次のexecパラメータをサポートしています。

- commandLine
- component
- id
- workingDir

開発ファイルイベント

現在、 は devfile のpostStartイベント CodeCatalyst のみをサポートしています。詳細については、Devfile.io ドキュメントpostStartObjectの「」を参照してください。

次の例は、devfile にpostStartイベントバインディングを追加する方法を示しています。

```
commands:
  - id: executescript
    exec:
      component: test
      commandLine: "./projects/devfiles/script.sh"
  - id: updateyum
    exec:
      component: test
      commandLine: "yum -y update --security"
```

```
events:
  postStart:
    - updateyum
    - executescript
```

起動後、開発環境は、定義された順序で指定されたpostStartコマンドを実行します。コマンドが失敗した場合、開発環境は引き続き実行され、実行出力は のログに保存されます/aws/mde/logs。

Devfile コンポーネント

現在、 は devfile 内のcontainerコンポーネント CodeCatalyst のみをサポートしています。詳細については、Devfile.io ドキュメントの「コンポーネントの追加」を参照してください。

次の例は、devfile のコンテナにスタートアップコマンドを追加する方法を示しています。

```
components:
  - name: test
    container:
      image: public.ecr.aws/amazonlinux/amazonlinux:2
      command: ['sleep', 'infinity']
```

> **Note**
>
> コンテナの存続期間が短いエントリコマンドがある場合は、 を含めてコンテナを実行command: ['sleep', 'infinity']し続ける必要があります。

CodeCatalyst は、コンテナコンポーネントで次のプロパティもサポートしています: args、env、mountSources、volumeMounts。

VPC 接続を開発環境に関連付ける

VPC 接続は、ワークフローが VPC にアクセスするために必要なすべての設定を含む CodeCatalyst リソースです。スペース管理者は、スペースメンバーに代わって Amazon CodeCatalyst コンソールに独自の VPC 接続を追加できます。VPC 接続を追加することで、スペースメンバーはワークフローアクションを実行し、ネットワークルールに準拠し、関連付けられた VPC 内のリソースにアクセスできる開発環境を作成できます。

開発環境は、開発環境の作成時にのみ VPC 接続に関連付けることができます。開発環境に関連付けられた VPC 接続は、作成後に変更することはできません。別の VPC 接続を使用する場合は、現在の開発環境を削除して新しい開発環境を作成する必要があります。

> **Important**
>
> VPC 接続の開発環境は、 にリンクされたサードパーティーのソースリポジトリ CodeCatalystをサポートしていません。

開発環境では、作成時に複数の AWS リソースとサービスが使用されることに注意してください。つまり、開発環境は次の AWS サービスに接続します。

- Amazon CodeCatalyst
- AWS SSM
- AWS KMS
- Amazon ECR
- Amazon CloudWatch
- Amazon ECS

> **Note**
>
> AWS Toolkit は、関連付けられた VPC 接続を使用した開発環境の作成をサポートしていません。また、 以外の IDE を使用すると AWS Cloud9、約 5 分のロード時間が発生する可能性があることに注意してください。

スペースレベルで VPC 接続を管理するには、スペース管理者ロールまたはパワーユーザーロールが必要です。VPC の詳細については、「 管理者ガイド」のVPCs の管理 CodeCatalyst」を参照してください。 CodeCatalyst

の開発環境のクォータ CodeCatalyst

次の表に、Amazon の開発環境のクォータと制限を示します CodeCatalyst。Amazon のクォータの詳細については CodeCatalyst、「」を参照してくださいのクォータ CodeCatalyst。

1 か月あたりの開発環境時間数	開発環境の時間は、スペース全体のストレージ制限の影響を受けます。詳細については、「料金表」と「開発環境の問題のトラブルシューティング」を参照してください。
スペースあたりの開発環境ストレージの量	開発環境ストレージは、スペース全体のストレージ制限の影響を受けます。詳細については、「料金表」と「開発環境の問題のトラブルシューティング」を参照してください。
開発環境のコンピューティング量	開発環境のコンピューティングは、スペース全体のストレージ制限の影響を受けます。詳細については、「料金表」と「開発環境の問題のトラブルシューティング」を参照してください。

でソフトウェアパッケージを公開および共有する CodeCatalyst

Amazon CodeCatalyst には、アプリケーション開発に使用されるソフトウェアパッケージを安全に保存および共有できるフルマネージドパッケージリポジトリサービスが含まれています。これらのパッケージはパッケージリポジトリに保存され、 のプロジェクト内で作成および整理されます CodeCatalyst。

CodeCatalyst では、次のパッケージ形式がサポートされています。

- npm

パッケージリポジトリ内のパッケージは、リポジトリを含むプロジェクトのメンバー間で検出および共有できます。

リポジトリにパッケージを追加するには、リポジトリエンドポイント (URL) を使用するようにパッケージマネージャーを設定します。その後、パッケージマネージャーを使用して、パッケージをリポジトリに公開できます。

パッケージリポジトリにパッケージを発行し、 CodeCatalyst パッケージリポジトリからパッケージを使用する CodeCatalyst ようにワークフローを設定できます。ワークフローでのパッケージの使用の詳細については、「」を参照してくださいワークフローを使用したパッケージの公開とインポート。

アップストリームリポジトリとして追加することで、あるパッケージリポジトリのパッケージを同じプロジェクトの別のリポジトリで使用できるようにすることができます。アップストリームリポジトリで使用可能なすべてのパッケージバージョンは、ダウンストリームリポジトリでも使用できます。

パブリックな外部 CodeCatalyst リポジトリをリポジトリに接続することで、リポジトリでオープンソースパッケージを使用できるようになります。サポートされているリポジトリのリストなど、アップストリームリポジトリと外部リポジトリへの接続の詳細については、「」を参照してくださいアップストリームリポジトリの設定と使用。

トピック

- パッケージの概念
- パッケージリポジトリの設定と使用
- アップストリームリポジトリの設定と使用

- パブリック外部リポジトリへの接続
- パッケージの公開と変更
- npmを使う
- パッケージのクォータ

パッケージの概念

でパッケージを管理、公開、または使用する際に知っておくべき概念と用語をいくつか紹介しますCodeCatalyst。

パッケージ

パッケージは、ソフトウェアのインストールと依存関係の解決に必要なソフトウェアとメタデータの両方を含むバンドルです。 は npm パッケージ形式 CodeCatalyst をサポートします。

パッケージは以下で構成されます。

- 名前 (例えば、 webpackは一般的な npm パッケージの名前)
- オプションの 名前空間 (など@types@types/node)
- 一連のバージョン (例: 、1.0.01.0.1、1.0.2)
- パッケージレベルのメタデータ (npm dist タグなど)

パッケージ名前空間

一部のパッケージ形式は、階層的なパッケージ名をサポートしてパッケージを論理グループに整理し、名前の衝突を回避します。同じ名前のパッケージは、異なる名前空間に保存できます。例えば、npm はスコープをサポートし、npm パッケージのスコープ@types/nodeは @types、名前はですnode。他にも多くのパッケージ名が@typesスコープにあります。では CodeCatalyst、スコープ (「タイプ」) はパッケージ名前空間と呼ばれ、名前 (「ノード」) はパッケージ名と呼ばれます。パッケージ名をグループ化する方法がない場合は、名前の競合を回避するのがより難しい場合があります。

パッケージバージョン

パッケージバージョンは、@types/node@12.6.9のようにパッケージの特定のバージョンを識別します。バージョン番号の形式とセマンティクスは、パッケージ形式によって異なります。例え

ば、npmパッケージのバージョンはセマンティックバージョニングの仕様 に準拠する必要があります。では CodeCatalyst、パッケージバージョンはバージョン識別子、 package-version-level メタデータ、およびアセットのセットで構成されます。

アセット

アセットは、npm ファイルなど、パッケージバージョンに関連付けられている に保存されている個々の.tgzファイル CodeCatalyst です。

パッケージリポジトリ

CodeCatalyst パッケージリポジトリには、パッケージバージョン を含む一連のパッケージ が含まれており、それぞれが一連のアセット にマッピングされます。各パッケージリポジトリは、Node.js CLI (npm) などのツールを使用してパッケージを取得および公開するためのエンドポイントを提供します。各スペースに最大 1,000 個のパッケージリポジトリを作成できます。

アップストリームリポジトリを使用して、パッケージリポジトリを別のリポジトリにリンクできます。パッケージリポジトリをアップストリームリポジトリとしてリンクすると、設定されたリポジトリを介してリンクされたリポジトリ内のパッケージを使用できます。詳細については、「アップストリームリポジトリ」を参照してください。

ゲートウェイリポジトリは、公式の外部パッケージ機関からパッケージをプルして保存する特殊なタイプのパッケージリポジトリです。詳細については、「ゲートウェイリポジトリ」を参照してください。

ゲートウェイリポジトリ

ゲートウェイリポジトリは、サポートされている外部公式パッケージ権限に接続されている特殊なタイプのパッケージリポジトリです。ゲートウェイリポジトリをアップストリームリポジトリ として追加すると、対応する公式パッケージ権限のパッケージを使用できます。ダウンストリームリポジトリはパブリックリポジトリと通信せず、すべてゲートウェイリポジトリによって仲介されます。この方法で消費されたパッケージは、ゲートウェイリポジトリと、元のリクエストを受信したダウンストリームリポジトリの両方に保存されます。

ゲートウェイリポジトリは事前定義されていますが、使用する各プロジェクトで作成する必要があります。次のリストには、 で作成できるすべてのゲートウェイリポジトリ CodeCatalyst と、それらが接続されているパッケージ権限が含まれています。

- npm-public-registry-gateway は、npmjs.com から npm パッケージを提供します。

アップストリームリポジトリ

を使用して CodeCatalyst 、2 つのパッケージリポジトリ間にアップストリーム関係を作成できます。パッケージリポジトリは、含まれるパッケージバージョンにダウンストリームリポジトリのパッケージリポジトリエンドポイントからアクセスできる場合、別の のアップストリームです。アップストリーム関係では、2 つのパッケージリポジトリの内容は、クライアントの観点から効果的にマージされます。

例えば、パッケージマネージャーがリポジトリに存在しないパッケージバージョンをリクエストすると、 CodeCatalyst は設定されたアップストリームリポジトリでパッケージバージョンを検索します。アップストリームリポジトリは設定された順序で検索され、パッケージが見つかると検索 CodeCatalyst が停止します。

パッケージリポジトリの設定と使用

では CodeCatalyst、パッケージはパッケージリポジトリ内に保存および管理されます。パッケージを に公開 CodeCatalyst したり、 CodeCatalyst （ またはサポートされているパブリックパッケージリポジトリ) からパッケージを消費したりするには、パッケージリポジトリを作成し、パッケージマネージャーをそのリポジトリに接続する必要があります。

トピック

パッケージリポジトリの作成

でパッケージリポジトリを作成するには、次のステップを実行します CodeCatalyst。

パッケージリポジトリを作成するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. パッケージリポジトリを作成するプロジェクトに移動します。
3. ナビゲーションペインから、パッケージ を選択します。
4. 「パッケージリポジトリ」ページで、「パッケージリポジトリの作成」を選択します。

5. 「パッケージリポジトリの詳細」セクションで、以下を追加します。

   a. リポジトリ名 。プロジェクト名やチーム名、リポジトリの使用方法などの詳細を含むわかりやすい名前を使用することを検討してください。

   b. (オプション) リポジトリの説明 。リポジトリの説明は、プロジェクト内の複数のチームに複数のリポジトリがある場合に特に役立ちます。

6. 「アップストリームリポジトリの編集」セクションで、パッケージリポジトリを介してアクセスする CodeCatalyst パッケージリポジトリを追加します。Gateway リポジトリを追加して、外部パッケージリポジトリまたは他の CodeCatalyst パッケージリポジトリに接続できます。

   - パッケージがパッケージリポジトリからリクエストされると、アップストリームリポジトリはこのリストに表示される順序で検索されます。パッケージが見つかると、 CodeCatalyst は検索を停止します。アップストリームリポジトリの順序を変更するには、リスト内のリポジトリをドラッグアンドドロップするか、並べ替えボタンを使用します。

7. 作成 を選択して、パッケージリポジトリを作成します。

パッケージリポジトリへの接続

にパッケージを発行 CodeCatalyst したり、 から使用したりするには CodeCatalyst、パッケージリポジトリのエンドポイント情報と CodeCatalyst 認証情報を使用してパッケージマネージャーを設定する必要があります。リポジトリを作成していない場合は、「」の手順に従って作成できますパッケージリポジトリの作成。

npm パッケージマネージャーをパッケージリポジトリに接続する方法については、 CodeCatalyst「」を参照してくださいnpm の設定と使用。

パッケージリポジトリの編集

パッケージリポジトリの説明とアップストリームリポジトリを編集するには、次のステップを実行します。

パッケージリポジトリを編集するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 編集するパッケージリポジトリを含むプロジェクトに移動します。
3. ナビゲーションペインから、パッケージ を選択します。
4. パッケージリポジトリページで、削除するリポジトリを選択します。

5. アクションドロップダウンを選択し、編集 を選択します。
6. リポジトリの説明とアップストリームリポジトリを編集します。アップストリームリポジトリの作成方法の詳細については、「アップストリームリポジトリの設定と使用」を参照してください。
7. [保存] を選択します。

パッケージリポジトリの削除

でパッケージリポジトリを削除するには、次のステップを実行します CodeCatalyst。

パッケージリポジトリを削除するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 削除するパッケージリポジトリを含むプロジェクトに移動します。
3. ナビゲーションペインから、パッケージ を選択します。
4. パッケージリポジトリページで、削除するリポジトリを選択します。
5. Actions ドロップダウンを選択し、Delete を選択します。
6. パッケージリポジトリを削除した場合の影響について、提供された情報を確認します。
7. 入力フィールドに と入力し、`delete`「削除」を選択します。

アップストリームリポジトリの設定と使用

ゲートウェイリポジトリと他のパッケージリポジトリの両方を、アップストリームとしてパッケージリポジトリに接続できます。これにより、パッケージマネージャークライアントは、1 つのパッケージリポジトリエンドポイントを使用して、複数のパッケージリポジトリに含まれるパッケージにアクセスできます。アップストリームリポジトリを使用する主な利点は次のとおりです。

- 複数のソースからプルするには、1 つのリポジトリエンドポイントでパッケージマネージャーを設定するだけで済みます。
- アップストリームリポジトリから消費されるパッケージはダウンストリームリポジトリに保存されるため、アップストリームリポジトリで予期しない停止が発生した場合でもパッケージを使用できます。

パッケージリポジトリを作成するときに、アップストリームリポジトリを追加できます。CodeCatalyst コンソールの既存のパッケージリポジトリからアップストリームリポジトリを追加または削除することもできます。

ゲートウェイリポジトリをアップストリームリポジトリとして追加すると、パッケージリポジトリはゲートウェイリポジトリの対応するパブリックパッケージリポジトリに接続されます。サポートされているパブリックパッケージリポジトリのリストについては、「」を参照してくださいサポートされている外部パッケージリポジトリとそのゲートウェイリポジトリ。

複数のリポジトリをアップストリームリポジトリとしてリンクできます。例えば、チームが という名前のリポジトリを作成し、 team-repoをアップストリームリポジトリとしてnpm-public-registry-gateway追加project-repoした という名前の別のリポジトリを既に使用していて、パブリック npm リポジトリ に接続されているとしますnpmjs.com。をアップストリームリポジトリteam-repoとして に追加できますproject-repo。この場合、、、、team-reponpm-public-registry-gatewayおよび からパッケージをプルproject-repoするために project-repoを使用するようにパッケージマネージャーを設定するだけで済みますnpmjs.com。

トピック

- アップストリームリポジトリの追加
- アップストリームリポジトリの検索順序の編集
- アップストリームリポジトリを持つパッケージバージョンのリクエスト
- アップストリームリポジトリの削除

アップストリームリポジトリの追加

パブリックパッケージリポジトリまたは別の CodeCatalyst パッケージリポジトリをアップストリームリポジトリとしてダウンストリームリポジトリに追加すると、アップストリームリポジトリ内のすべてのパッケージが、ダウンストリームリポジトリに接続されているパッケージマネージャーで利用できるようになります。

アップストリームリポジトリを追加するには

1. ナビゲーションペインで、[Packages (パッケージ)] を選択します。
2. 「パッケージリポジトリ」ページで、アップストリームリポジトリを追加するパッケージリポジトリを選択します。
3. アクションドロップダウンメニューを選択し、編集 を選択します。

4. 「アップストリームリポジトリ」セクションで、「アップストリームリポジトリを追加」を選択します。
5. 検索バーを選択して、使用可能なリポジトリのリストを表示して検索します。サポートされているパブリックパッケージリポジトリまたは他の CodeCatalyst リポジトリをアップストリームリポジトリとして追加できます。追加するリポジトリが見つかったら、リストから選択します。

> **Note**
>
> Gateway リポジトリ には、 npm-public-registry-gatewayリポジトリがあります。npmjs.com などのパブリック外部パッケージ機関に接続する CodeCatalyst には、ゲートウェイリポジトリを、外部リポジトリからプルされたパッケージを検索して保存する中間リポジトリとして使用します。これにより、プロジェクト内のすべてのパッケージリポジトリがゲートウェイリポジトリのパッケージを使用するため、時間とデータ転送が節約されます。

6. アップストリームリポジトリとして追加するすべてのリポジトリを選択したら、「追加」を選択します。
7. アップストリームリポジトリの検索順序の変更の詳細については、「」を参照してくださいアップストリームリポジトリの検索順序の編集。

アップストリームリポジトリを追加したら、ローカルリポジトリに接続されているパッケージマネージャーを使用して、アップストリームリポジトリからパッケージを取得できます。パッケージマネージャーの設定を更新する必要はありません。アップストリームリポジトリからのパッケージバージョンのリクエストの詳細については、「」を参照してくださいアップストリームリポジトリを持つパッケージバージョンのリクエスト。

アップストリームリポジトリの検索順序の編集

CodeCatalyst は、設定された検索順序でアップストリームリポジトリを検索します。パッケージが見つかると、 は検索を CodeCatalyst 停止します。アップストリームリポジトリがパッケージを検索する順序を変更できます。

アップストリームリポジトリの検索順序を編集するには

1. ナビゲーションペインで、[Packages (パッケージ)] を選択します。
2. リポジトリページで、アップストリームリポジトリの検索順序を編集するパッケージリポジトリを選択します。

3. アクションドロップダウンを選択し、編集 を選択します。
4. アップストリームリポジトリセクションでは、アップストリームリポジトリとその検索順序を表示できます。検索順序を変更するには、リスト内のリポジトリをドラッグアンドドロップするか、並べ替えボタンを使用します。
5. アップストリームリポジトリの検索順序の編集が完了したら、保存 を選択します。

アップストリームリポジトリを持つパッケージバージョンのリクエスト

次の例は、パッケージマネージャーがアップストリームリポジトリを持つパッケージリポジトリからパッケージを CodeCatalyst リクエストする場合のシナリオを示しています。

この例では、 などのパッケージマネージャーは、複数のアップストリームリポジトリを持つ という名前のパッケージリポジトリからパッケージバージョンdownstreamをnpmリクエストします。パッケージがリクエストされると、次のことが発生する可能性があります。

- リクエストされたパッケージバージョンがdownstreamに含まれる場合、クライアントにリターンされます。
- にリクエストされたパッケージバージョンが含まれdownstreamていない場合、 は設定された検索順序で downstreamのアップストリームリポジトリでそのバージョン CodeCatalyst を検索します。パッケージバージョンが見つかると、そのバージョンへのリファレンスがdownstreamにコピーされます、そしてパッケージのバージョンがクライアントに返されます。
- downstream またはアップストリームリポジトリにパッケージバージョンが含まれていない場合、HTTP 404 Not Foundレスポンスがクライアントに返されます。

1 つのリポジトリに許可される直接アップストリームリポジトリの最大数は 10 です。パッケージバージョンがリクエストされたときに で CodeCatalyst 検索されるリポジトリの最大数は 25 です。

アップストリームリポジトリからのパッケージの保持

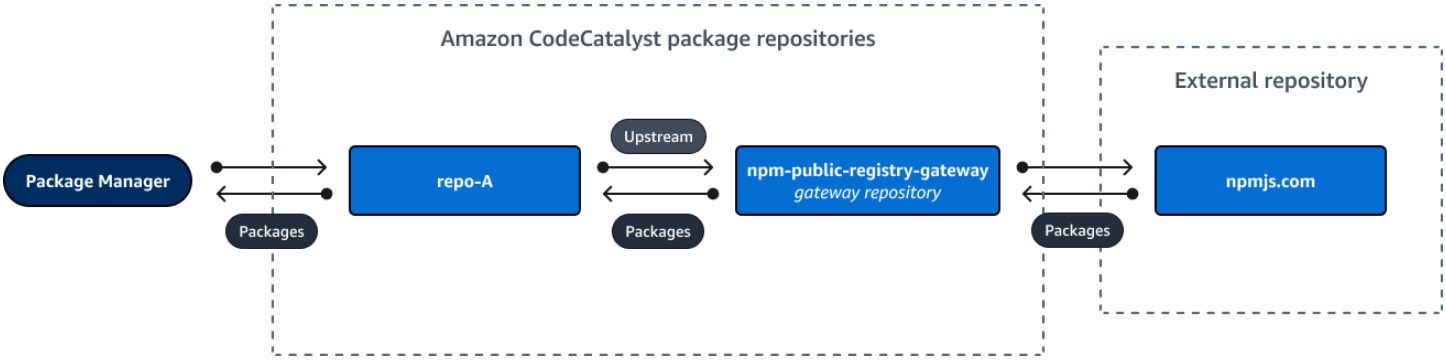
リクエストされたパッケージバージョンがアップストリームリポジトリで見つかった場合、そのバージョンへの参照は保持され、リクエストしたリポジトリで常に使用できます。これにより、アップストリームリポジトリが予期せず停止した場合、パッケージにアクセスできます。保持されたパッケージバージョンは、次のいずれの影響も受けません:

- アップストリームリポジトリの削除。
- アップストリームリポジトリのダウンストリームリポジトリからの切断。
- アップストリームリポジトリからのパッケージバージョンの削除。

- アップストリームリポジトリのパッケージバージョンの編集 (例えば、新しいアセットを追加するなど) 。

アップストリームリレーションシップによるパッケージの取得

CodeCatalyst は、アップストリームリポジトリと呼ばれる複数のリンクされたリポジトリを介してパッケージを取得できます。 CodeCatalyst パッケージリポジトリがゲートウェイリポジトリへのアップストリーム接続を持つ別の CodeCatalyst パッケージリポジトリへのアップストリーム接続を持つ場合、アップストリームリポジトリにないパッケージのリクエストは外部リポジトリからコピーされます。例えば、 という名前のリポジトリrepo-Aがゲートウェイリポジトリ へのアップストリーム接続を持つとしますnpm-public-registry-gateway。 npm-public-registry-gatewayはパブリックパッケージリポジトリ https://npmjs.com へのアップストリーム接続を持つとします。



repo-A がリポジトリを使用するようにnpm設定されている場合、 を実行すると、https://npmjs.com から へのパッケージのコピーnpm installが開始されますnpm-public-registry-gateway。インストールされているバージョンもrepo-Aプルされます。次の例では、lodashがインストールされます。

```
$ npm config get registry
https://packages.region.codecatalyst.aws/npm/space-name/proj-name/repo-name/
$ npm install lodash
+ lodash@4.17.20
added 1 package from 2 contributors in 6.933s
```

の実行後npm install、 には最新バージョン (lodash 4.17.20) のみrepo-Aが含まれます。これは、 npmから によって取得されたバージョンであるためですrepo-A。

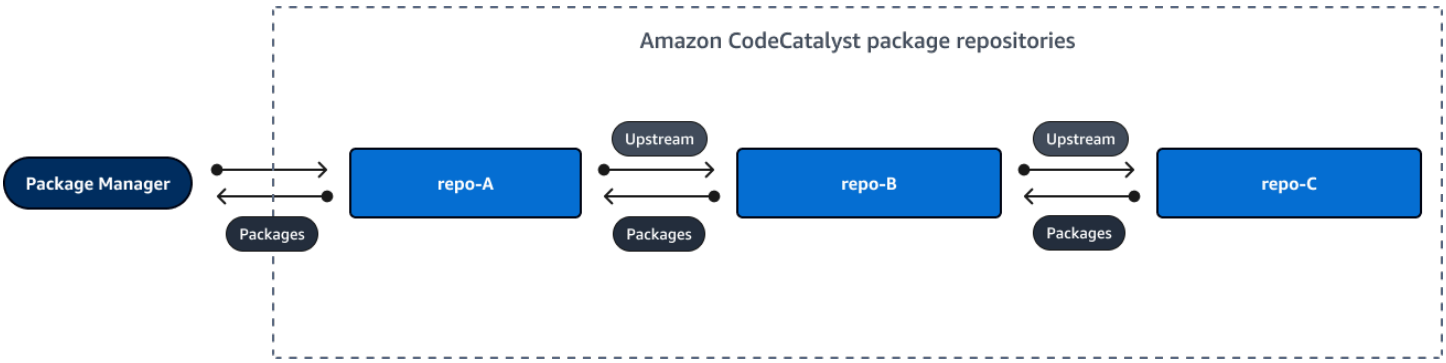
npm-public-registry-gateway には https://npmjs.com への外部アップストリーム接続があるため、https://npmjs.com からインポートされたすべてのパッケージバージョンは に保存されますnpm-

public-registry-gateway。これらのパッケージバージョンは、 へのアップストリーム接続を持つダウンストリームリポジトリによって取得された可能性がありますnpm-public-registry-gateway。

の内容は、https://npmjs.com からインポートされたすべてのパッケージとパッケージバージョンを経時的に表示する方法npm-public-registry-gatewayを提供します。

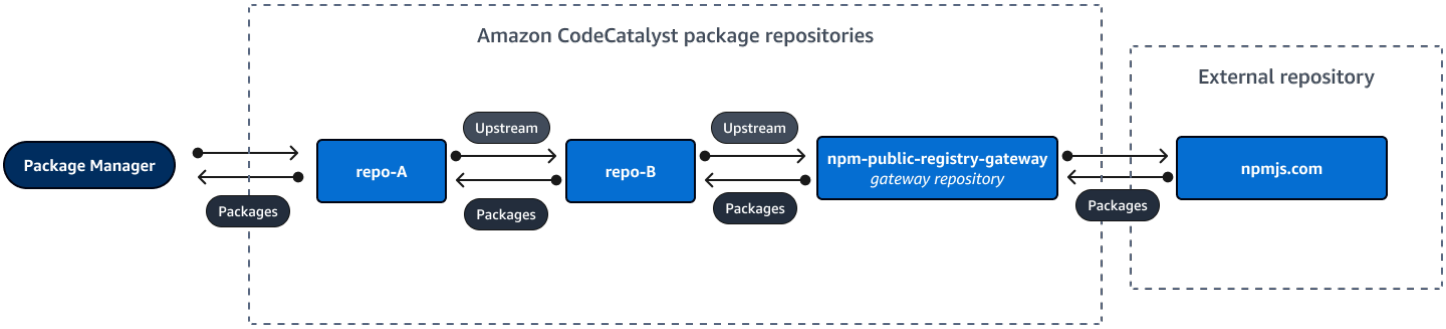
中間リポジトリでのパッケージの保持

CodeCatalyst では、アップストリームリポジトリを連鎖できます。例えば、 repo-Aはアップストリームリポジトリrepo-Bとして、 repo-Bはアップストリームリポジトリrepo-Cとして使用できます。この設定により、repo-Bとrepo-Cにあるパッケージバージョンがrepo-Aから入手可能になります。



パッケージマネージャーがリポジトリに接続repo-Aし、リポジトリ からパッケージバージョンを取得するとrepo-C、パッケージバージョンはリポジトリ に保持されませんrepo-B。パッケージバージョンは、最も遠いダウンストリームリポジトリにのみ保持されます。この例では ですrepo-A。中間リポジトリには保持されません。これは、長いチェーンにも当てはまります。例えば、、、repo-A、repo-Brepo-Cおよび の 4 つのリポジトリがありrepo-D、 repo-A からパッケージバージョンを取得するために接続されているパッケージマネージャーがある場合、パッケージバージョンは に保持されますrepo-Aがrepo-D、 repo-Bまたは には保持されませんrepo-C。

パッケージの保持動作は、パブリックパッケージリポジトリからパッケージバージョンをプルする場合も同様です。ただし、パッケージバージョンは常にパブリックリポジトリへの直接アップストリーム接続を持つゲートウェイリポジトリに保持されます。例えば、 repo-Aはアップストリームリポジトリrepo-Bとして を持ちます。 repo-B はアップストリームリポジトリnpm-public-registry-gatewayとして を持ち、パブリックリポジトリ npmjs.com へのアップストリーム接続があります。以下の図を参照してください。



に接続されたパッケージマネージャーが特定のパッケージバージョン、例えば lodash 4.17.20 をrepo-Aリクエストし、そのパッケージバージョンが 3 つのリポジトリのいずれにも存在しない場合、npmjs.com から取得されます。lodash 4.17.20 がフェッチされると、最も遠いダウンストリームリポジトリrepo-Aとして に保持され、パブリック外部リポジトリ npm-public-registry-gateway npmjs.com へのアップストリーム接続があります。lodash 4.17.20 は中間リポジトリrepo-Bであるため、 には保持されません。

アップストリームリポジトリの削除

アップストリームリポジトリ内のパッケージにアクセスする必要がなくなった場合は、アップストリームリポジトリをパッケージリポジトリから削除できます。

> **Warning**
>
> アップストリームリポジトリを削除すると、アップストリームのリレーションシップチェーンが壊れ、プロジェクトやビルドが破損する可能性があります。

アップストリームリポジトリを削除するには

1. ナビゲーションペインで、[Packages (パッケージ)] を選択します。
2. 「パッケージリポジトリ」ページで、アップストリームリポジトリを削除するパッケージリポジトリを選択します。
3. アクションドロップダウンを選択し、編集 を選択します。
4. 「アップストリームリポジトリ」セクションで、削除するアップストリームリポジトリを見つけ、「 の削除」を選択します。
5. アップストリームリポジトリの削除が完了したら、保存 を選択します。

パブリック外部リポジトリへの接続

対応するゲートウェイリポジトリをアップストリームリポジトリとして追加することで、CodeCatalyst パッケージリポジトリをサポートされているパブリック外部リポジトリに接続できます。ゲートウェイリポジトリは、外部リポジトリからプルされたパッケージを検索して保存する中間リポジトリとして機能します。これにより、プロジェクト内のすべてのパッケージリポジトリがゲートウェイリポジトリのパッケージを使用するため、時間とデータ転送が節約されます。

ゲートウェイリポジトリを使用してパブリックリポジトリに接続するには

1. ナビゲーションペインで、[Packages (パッケージ)] を選択します。
2. 「パッケージリポジトリ」ページの「ゲートウェイリポジトリ」で、サポートされているゲートウェイリポジトリのリストとその説明を表示できます。ゲートウェイリポジトリを使用するには、まずゲートウェイリポジトリを作成する必要があります。ゲートウェイリポジトリが作成されている場合は、作成された日時が表示されます。まだ作成していない場合は、作成 を選択して作成します。
3. パブリックリポジトリに接続するパッケージリポジトリを選択します。
4. Actions ドロップダウンメニューを選択し、編集 を選択します。
5. パブリックリポジトリに接続するには、アップストリームリポジトリとして接続するパブリックリポジトリに対応するゲートウェイリポジトリを追加します。

   「アップストリームリポジトリの編集」セクションで、 CodeCatalyst 「リポジトリの追加」を選択します。
6. Gateway リポジトリセクションには、使用可能なすべてのゲートウェイリポジトリが一覧表示されます。接続するパブリックの外部リポジトリに対応するゲートウェイリポジトリを見つけたら、リストからそのリポジトリを選択し、「追加」を選択します。
7. パッケージがリポジトリからリクエストされると、 はアップストリームリポジトリの編集リストに表示される順序でアップストリームリポジトリ CodeCatalyst を検索します。パッケージが見つかると、 は検索を CodeCatalyst 停止します。アップストリームリポジトリの順序を変更するには、リポジトリをリストにドラッグアンドドロップするか、並べ替え矢印を使用します。
8. アップストリームリポジトリの追加と注文が完了したら、保存 を選択します。

ゲートウェイリポジトリをアップストリームリポジトリとして追加すると、ローカルリポジトリに接続されているパッケージマネージャーを使用して、それに対応するパブリックな外部パッケージリポジトリからパッケージを取得できます。パッケージマネージャーの設定を更新する必要はありません。この方法で消費されるパッケージは、ゲートウェイリポジトリとローカルパッケージリポジトリ

の両方に保存されます。アップストリームリポジトリからのパッケージバージョンのリクエストの詳細については、「」を参照してくださいアップストリームリポジトリを持つパッケージバージョンのリクエスト。

サポートされている外部パッケージリポジトリとそのゲートウェイリポジトリ

CodeCatalyst は、ゲートウェイリポジトリを使用して、以下の公式パッケージ権限へのアップストリーム接続の追加をサポートします。

リポジトリパッケージタイプ	説明	ゲートウェイリポジトリ名
npm	npm 公開レジストリ	`npm-public-registry-gateway`

パッケージの公開と変更

のパッケージ CodeCatalyst は、依存関係を解決し、ソフトウェアをインストールするために必要なソフトウェアとメタデータのバンドルです。 は npm パッケージ形式 CodeCatalyst をサポートしています。このセクションでは、パッケージの公開、表示、削除、パッケージバージョンのステータスの更新について説明します。

トピック

- パッケージリポジトリへの CodeCatalyst パッケージの発行
- パッケージバージョンの詳細の表示
- パッケージバージョンの削除
- パッケージバージョンのステータスの更新
- パッケージオリジンコントロールの編集

パッケージリポジトリへの CodeCatalyst パッケージの発行

パッケージマネージャーツールを使用して、サポートされている任意の CodeCatalyst パッケージタイプのバージョンをパッケージリポジトリに発行できます。パッケージバージョンを発行する手順は次のとおりです。

パッケージバージョンを CodeCatalyst パッケージリポジトリに発行するには

1. まだ作成していない場合は、パッケージリポジトリ を作成します。
2. パッケージマネージャーをパッケージリポジトリに接続します。npm パッケージマネージャーをパッケージリポジトリに接続する方法については、 CodeCatalyst 「」を参照してくださいnpm の設定と使用。
3. 接続されたパッケージマネージャーを使用して、パッケージバージョンを公開します。

目次

公開リポジトリとアップストリームリポジトリ

では CodeCatalyst、到達可能なアップストリームリポジトリまたはパブリックリポジトリに存在するパッケージバージョンを公開することはできません。例えば、npm パッケージ をパッケージリポジトリ `lodash@1.0`に公開するとします。 `myrepo`myrepoは、アップストリームリポジトリとして設定されたゲートウェイリポジトリを介して npmjs.com に接続されます。`lodash@1.0` がアップストリームリポジトリまたは npmjs.com に存在する場合、 は 409 競合エラーを発行`myrepo`して、 でそのリポジトリに発行しようとする試み CodeCatalyst を拒否します。これにより、アップストリームリポジトリのパッケージと同じ名前とバージョンのパッケージを誤って公開するのを防ぐことができ、予期しない動作が発生する可能性があります。

アップストリームリポジトリに存在するパッケージ名の異なるバージョンを公開することはできます。たとえば、`lodash@1.0`はアップストリームのリポジトリに存在しますが、`lodash@1.1`がそうではない場合、`lodash@1.1`をダウンストリームのリポジトリで公開します。

プライベートパッケージと公開リポジトリ

CodeCatalyst は、 CodeCatalyst リポジトリに保存されているパッケージをパブリックリポジトリに公開しません。例えば、npmjs.com. CodeCatalyst imports パッケージをパブリックリポジトリから CodeCatalyst リポジトリに公開しますが、パッケージを逆方向に移動させることはありません。 CodeCatalyst リポジトリに発行するパッケージはプライベートのままで、リポジトリが属する CodeCatalyst プロジェクトでのみ使用できます。

パッケージアセットの上書き

含まれているコンテンツが異なるパッケージアセットを既に存在しているパッケージアセットを再公開することはできません。npm はパッケージバージョンごとに 1 つのアセットしかサポートしていないため、公開されたパッケージバージョンを変更するには、まずアセットを削除する必要があります。

パッケージバージョンの詳細の表示

CodeCatalyst コンソールを使用して、特定のパッケージバージョンの詳細を表示できます。

パッケージバージョンの詳細を表示するには

1. ナビゲーションペインで、[Packages (パッケージ)] を選択します。
2. パッケージリポジトリページで、詳細を表示するパッケージバージョンを含むリポジトリを選択します。
3. Packages テーブルでパッケージバージョンを検索します。検索バーを使用して、パッケージ名でパッケージをフィルタリングできます。リストからパッケージを選択します。
4. パッケージの詳細ページで、バージョン を選択し、表示するバージョンを選択します。

パッケージバージョンの削除

CodeCatalyst コンソールのパッケージバージョンの詳細ページからパッケージバージョンを削除できます。

パッケージバージョンを削除するには

1. ナビゲーションペインで、[Packages (パッケージ)] を選択します。
2. 「パッケージリポジトリ」ページで、削除するパッケージバージョンを含むリポジトリを選択します。
3. テーブルからパッケージを検索して選択します。
4. パッケージの詳細ページで、バージョンを選択し、削除するバージョンを選択します。
5. パッケージバージョンの詳細ページで、バージョンアクションを選択し、削除を選択します。
6. テキストフィールドに delete と入力し、Delete を選択します。

パッケージバージョンのステータスの更新

のすべてのパッケージバージョン CodeCatalyst には、パッケージバージョンの現在の状態と可用性を記述するステータスがあります。コンソールでパッケージバージョンのステータスを CodeCatalyst変更できます。パッケージバージョンで指定できるステータス値とその意味の詳細については、「」を参照してくださいパッケージバージョンのステータス。

パッケージバージョンのステータスを更新するには

1. ナビゲーションペインで、[Packages (パッケージ)] を選択します。
2. パッケージリポジトリページで、ステータスを更新するパッケージバージョンを含むリポジトリを選択します。
3. テーブルからパッケージを検索して選択します。
4. パッケージの詳細ページで、バージョンを選択し、表示するバージョンを選択します。
5. パッケージバージョンの詳細ページで、アクションを選択し、リスト解除、アーカイブ、または の破棄を選択します。各パッケージバージョンのステータスについては、「」を参照してくださいパッケージバージョンのステータス。
6. テキストフィールドに確認テキストを入力し、更新先のステータスに応じて、リスト解除、アーカイブ、または破棄を選択します。

パッケージバージョンのステータス

パッケージバージョンのステータスに使用できる値は次のとおりです。コンソールでパッケージバージョンのステータスを変更できます。詳細については、「パッケージバージョンのステータスの更新」を参照してください。

- 公開済み : パッケージバージョンは正常に公開され、パッケージマネージャーによってリクエストできます。パッケージバージョンは、パッケージマネージャーに返されるパッケージバージョンリストに含まれます。例えば、 の出力に含まれますnpm view <package-name> versions。パッケージバージョンのすべてのアセットは、リポジトリから入手できます。
- リストにない : パッケージバージョンアセットはリポジトリからダウンロードできますが、パッケージバージョンはパッケージマネージャーに返されるバージョンのリストに含まれません。例えば、npm パッケージの場合、 の出力npm view <package-name> versionsにはパッケージバージョンは含まれません。つまり、npm 依存関係解決ロジックはパッケージバージョンを選択しません。これは、そのバージョンが利用可能なバージョンのリストに表示されないためです。た

だし、リストにないパッケージバージョンが npm package-lock.json ファイルで既に参照されている場合、 の実行時などにダウンロードしてインストールできますnpm ci。

- アーカイブ済み : パッケージバージョンのアセットはダウンロードできません。パッケージバージョンは、パッケージマネージャーによって返されるバージョンのリストには含まれません。アセットが使用できないため、クライアントによるパッケージバージョンの使用はブロックされます。アプリケーションビルドがアーカイブ済み に更新されたバージョンに依存している場合、パッケージバージョンがローカルにキャッシュされていない限り、ビルドは失敗します。アーカイブされたパッケージバージョンはリポジトリにまだ存在するため、パッケージマネージャーまたはビルドツールを使用してアーカイブされたパッケージバージョンを再公開することはできません。ただし、コンソールでパッケージバージョンのステータスを未リストまたは公開に戻すことができます。
- 破棄済み: パッケージバージョンがリストに表示されず、アセットをリポジトリからダウンロードすることはできません。廃棄とアーカイブの主な違いは、廃棄ステータスが の場合、パッケージバージョンのアセットは によって完全に削除されます CodeCatalyst。このため、パッケージバージョンを[開放済み] から[アーカイブ済み]、[一覧表示されていない]、または [公開] に移動することはできません。アセットが削除されたため、パッケージバージョンを使用できません。パッケージバージョンが「廃棄済み」とマークされている場合、パッケージアセットのストレージに対して課金されません。

上記のリストのステータスに加えて、パッケージバージョンも削除できます。削除されたパッケージバージョンはリポジトリにないため、パッケージマネージャーまたはビルドツールを使用して、そのパッケージバージョンを自由に再公開できます。

パッケージオリジンコントロールの編集

Amazon では CodeCatalyst、パッケージバージョンを直接公開するか、アップストリームリポジトリからプルダウンするか、外部のパブリックリポジトリから取り込むことで、パッケージバージョンをパッケージリポジトリに追加できます。パッケージのバージョンを直接公開してパブリックリポジトリから取り込むことで、パッケージのバージョンを追加することを許可すると、依存関係置換攻撃に対して脆弱になります。詳細については、「依存関係置換攻撃」を参照してください。依存関係置換攻撃から保護するには、リポジトリ内のパッケージのパッケージオリジンコントロールを設定して、そのパッケージのバージョンをリポジトリに追加する方法を制限します。

異なるパッケージの新しいバージョンを、直接公開などの内部ソースとパブリックリポジトリなどの外部ソースの両方から取得するようにパッケージオリジンコントロールを設定することを検討する必要があります。デフォルトでは、パッケージオリジンコントロールは、パッケージの最初のバージョンがリポジトリに追加された方法に基づいて設定されます。

パッケージオリジンコントロール設定

パッケージオリジンコントロールでは、パッケージバージョンをリポジトリに追加する方法を設定できます。以下のリストには、使用可能なパッケージオリジンコントロールの設定と値が含まれています。

公開

この設定は、パッケージマネージャーや類似のツールを使用してパッケージのバージョンをリポジトリに直接公開できるかどうかを設定します。

- 許可: パッケージバージョンを直接公開できます。
- ブロック: パッケージバージョンは直接公開できません。

アップストリーム

この設定は、パッケージマネージャーからのリクエストに応じて、パッケージバージョンを外部のパブリックリポジトリから取り込むことができるか、アップストリームリポジトリから保持できるかを設定します。

- 許可: 任意のパッケージバージョンは、アップストリーム CodeCatalyst リポジトリとして設定された他のリポジトリから保持することも、外部接続を持つパブリックソースから取り込むこともできます。
- ブロック: パッケージバージョンは、アップストリーム CodeCatalyst リポジトリとして設定された他のリポジトリから保持したり、外部接続でパブリックソースから取り込んだりすることはできません。

パッケージオリジンコントロールのデフォルト設定

パッケージのデフォルトのパッケージオリジンコントロールは、そのパッケージの最初のバージョンがパッケージリポジトリに追加された方法に基づきます。

- 最初のパッケージバージョンがパッケージマネージャーによって直接公開された場合、設定は [公開: 許可] と [アップストリーム: ブロック] になります。
- 最初のパッケージバージョンがパブリックソースから取り込まれた場合、設定は [公開: ブロック] と [アップストリーム: 許可] になります。

一般的なパッケージアクセスコントロールシナリオ

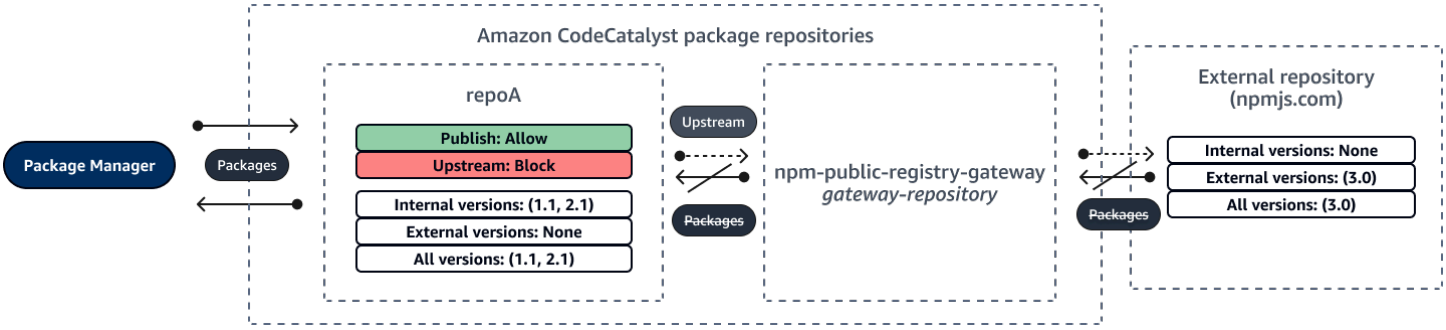
このセクションでは、パッケージバージョンが CodeCatalyst パッケージリポジトリに追加されるときの一般的なシナリオについて説明します。パッケージオリジンコントロールの設定は、最初のパッケージバージョンの追加方法に応じて、新しいパッケージに設定されます。

次のシナリオでは、内部パッケージがパッケージマネージャーから、維持するパッケージなどのリポジトリに直接公開されます。外部パッケージは、パブリックリポジトリに存在するパッケージで、外部接続でリポジトリに取り込むことができます。

外部パッケージバージョンが既存の内部パッケージに公開される

このシナリオでは、内部パッケージ「packageA」について考えてみます。チームは packageA の最初のパッケージバージョンを CodeCatalyst パッケージリポジトリに公開します。これはパッケージの最初のパッケージバージョンであるため、パッケージオリジンコントロール設定は自動的に [公開: 許可] および [アップストリーム: ブロック] に設定されます。パッケージがリポジトリで公開されると、同じ名前のパッケージが CodeCatalyst 、パッケージリポジトリに接続されているパブリックリポジトリに公開されます。これは、内部パッケージに対する依存関係置換攻撃の試みであったり、偶然であったりする可能性があります。いずれの場合でも、パッケージオリジンコントロールは、潜在的な攻撃からパッケージバージョンを保護するために、新しい外部バージョンの取り込みをブロックするように設定されています。

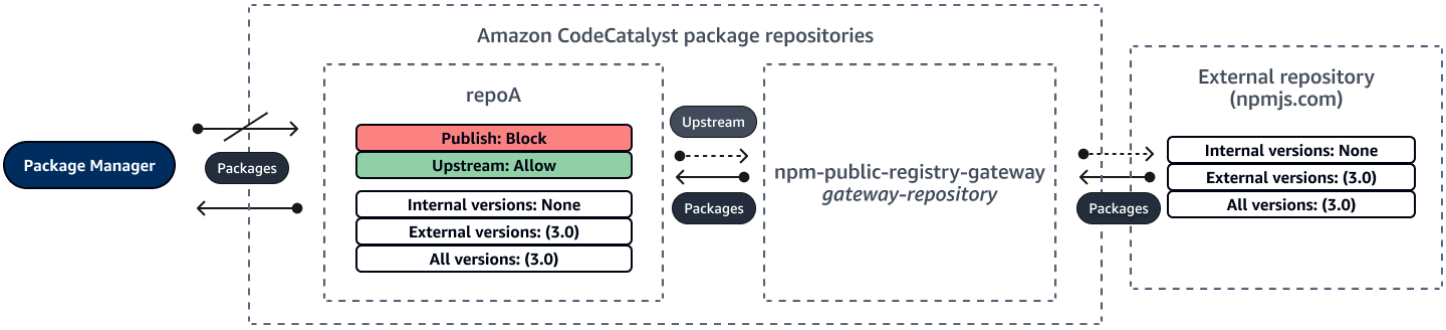
次のイメージでは、repoA はパブリックリポジトリへの外部接続を持つ CodeCatalyst パッケージリポジトリです。リポジトリには packageA のバージョン 1.1 と 2.1 が含まれていますが、バージョン 3.0 はパブリックリポジトリに公開されています。通常、repoA はパッケージがパッケージマネージャーによってリクエストされた後にバージョン 3.0 を取り込みます。パッケージの取り込みはブロック に設定されているため、バージョン 3.0 は CodeCatalyst パッケージリポジトリに取り込まれず、パッケージリポジトリに接続されているパッケージマネージャーでは使用できません。



内部パッケージバージョンが既存の外部パッケージに公開される

このシナリオでは、パッケージ packageBは、リポジトリに接続したパブリックリポジトリに外部で存在します。リポジトリに接続しているパッケージマネージャーが packageB をリクエストすると、パッケージバージョンはパブリックリポジトリからリポジトリに取り込まれます。これは packageB の最初のパッケージバージョンであるため、パッケージオリジンコントロール設定は自動的に [公開: ブロック] および [アップストリーム: 許可] に設定されます。その後、ユーザーは同じパッケージ名のバージョンをリポジトリに公開しようとします。パブリックパッケージを認識しておらず、同じ名前で無関係なパッケージを公開しようとしている場合や、パッチが適用されたバージョンを公開しようとしている場合、または既に外部に存在する正確なパッケージバージョンを直接公開しようとしている場合があります。 は公開しようとしているバージョン CodeCatalyst を拒否しますが、必要に応じて明示的に拒否を上書きしてバージョンを公開できます。

次のイメージでは、repoA はパブリックリポジトリへの外部接続を持つ CodeCatalyst パッケージリポジトリです。パッケージリポジトリには、パブリックリポジトリから取り込んだバージョン 3.0 が含まれています。バージョン 1.2 をパッケージリポジトリに公開したい。通常、バージョン 1.2 を repoAに発行することはできますが、発行がブロック に設定されているため、バージョン 1.2 は発行できません。



既存の外部パッケージにパッチを適用したパッケージバージョンを公開する

このシナリオでは、パッケージ packageBは、パッケージリポジトリに接続したパブリックリポジトリに外部で存在します。リポジトリに接続しているパッケージマネージャーが packageB をリクエストすると、パッケージバージョンはパブリックリポジトリからリポジトリに取り込まれます。これは packageB の最初のパッケージバージョンであるため、パッケージオリジンコントロール設定は自動的に [公開: ブロック] および [アップストリーム: 許可] に設定されます。チームは、このパッケージのパッチ適用されたパッケージバージョンをリポジトリに公開することにしました。パッケージバージョンを直接公開するために、チームはパッケージのオリジンコントロール設定を [公開: 許可] および アップストリーム: ブロック] に変更します。これで、このパッケージのバージョンをリポジトリに直接公開し、パブリックリポジトリから取り込むことができます。チームがパッチを適用したパッケージバージョンを公開した後、チームはパッケージオリジンの設定を [公開: ブロック] および [アップストリーム: 許可] に戻します。

パッケージオリジンコントロールの編集

パッケージオリジンコントロールは、パッケージの最初のパッケージバージョンがパッケージリポジトリに追加された方法に基づいて自動的に設定されます。詳細については、「パッケージオリジンコントロールのデフォルト設定」を参照してください。パッケージリポジトリ内のパッケージの CodeCatalyst パッケージオリジンコントロールを追加または編集するには、以下の手順を実行します。

パッケージオリジンコントロールを追加または編集するには

1. ナビゲーションペインで、[Packages (パッケージ)] を選択します。
2. 編集するパッケージを含むパッケージリポジトリを選択します。
3. パッケージ テーブルで、編集するパッケージを検索して選択します。
4. パッケージの概要ページから、オリジンコントロールの編集 を選択します。
5. オリジンコントロール で、このパッケージに設定するパッケージオリジンコントロールを選択します。パッケージオリジン制御設定である Publish と Upstream の両方を同時に設定する必要があります。

   - パッケージバージョンを直接公開できるようにするには、[公開] で [許可] を選択します。パッケージバージョンの公開を禁止するには、[ブロック] を選択します。
   - 外部リポジトリからのパッケージの取り込みとアップストリームリポジトリからのパッケージの取得を許可するには、[アップストリームソース] で [許可] を選択します。外部リポジトリおよびアップストリームリポジトリからのパッケージバージョンの取り込みとプルをすべてブロックするには、[ブロック] を選択します。
6. [保存] を選択します。

公開リポジトリとアップストリームリポジトリ

では CodeCatalyst、到達可能なアップストリームリポジトリまたはパブリックリポジトリに存在するパッケージバージョンを公開することはできません。例えば、npm パッケージをリポジトリ `lodash@1.0`に公開`myrepo`し、npmjs.com への外部接続を持つアップストリームリポジトリ`myrepo`があるとします。次のシナリオを考えてみます。

1. `lodash` 上のパッケージオリジンコントロール設定は、[公開: 許可] と [アップストリーム: 許可] です。`lodash@1.0` がアップストリームリポジトリまたは npmjs.com に存在する場合、 は 409 競合エラーを発行`myrepo`して、 でそのリポジトリに発行しようとする試み CodeCatalyst を拒否します。`lodash@1.1` などの別のバージョンを公開することもできます。

2. lodash 上のパッケージオリジンコントロール設定は、[公開: 許可] と [アップストリーム: ブロック] です。パッケージバージョンにアクセスできないため、まだ存在しない のバージョンをリポジトリlodashに公開できます。
3. lodash 上のパッケージオリジンコントロール設定は、[公開: ブロック] と [アップストリーム: 許可] です。この場合、どのパッケージバージョンもリポジトリに直接公開することはできません。

依存関係置換攻撃

パッケージマネージャーは、再利用可能なコードをパッケージ化して共有するプロセスを簡素化します。これらのパッケージは、ある組織がアプリケーションで使用するために開発したプライベートパッケージの場合もあれば、組織外で開発され、パブリックパッケージリポジトリによって配布されるパブリックパッケージ (通常はオープンソースパッケージ) の場合もあります。パッケージをリクエストする際、開発者はパッケージマネージャーを使用して依存関係の新しいバージョンを取得します。依存関係置換攻撃 (依存関係かく乱攻撃とも呼ばれる) は、通常、パッケージマネージャーでパッケージの正規バージョンと悪意のあるバージョンを区別できない点を悪用するものです。

依存関係置換攻撃は、ソフトウェアサプライチェーン攻撃と呼ばれる攻撃のサブセットに属します。ソフトウェアサプライチェーン攻撃は、ソフトウェアサプライチェーンのあらゆる場所にある脆弱性を利用する攻撃です。

依存関係置換攻撃は、内部で開発されたパッケージとパブリックリポジトリから取得したパッケージの両方を使用するすべてのユーザーを標的にする可能性があります。攻撃者は内部パッケージ名を特定し、同じ名前の悪意のあるコードを公開パッケージリポジトリに戦略的に配置します。通常、悪意のあるコードはバージョン番号の高いパッケージで公開されます。パッケージマネージャーは、悪意のあるパッケージをパッケージの最新バージョンとみなすため、これらの公開フィードから悪意のあるコードを取得します。これにより、目的のパッケージと悪意のあるパッケージの間に「混乱」または「置換」が発生し、コードが侵害されます。

依存関係置換攻撃を防ぐために、Amazon CodeCatalyst はパッケージオリジンコントロールを提供しています。パッケージオリジンコントロールは、パッケージをリポジトリに追加する方法を制御する設定です。コントロールは、新しいパッケージの最初のパッケージバージョンが CodeCatalyst リポジトリに追加されると自動的に設定されます。 コントロールを使用すると、パッケージバージョンをリポジトリに直接公開したり、パブリックソースから取り込んだりできないため、依存関係置換攻撃から保護できます。パッケージオリジンコントロールとその変更方法については、「パッケージオリジンコントロールの編集」を参照してください。

npmを使う

これらのトピックではnpm、 で Node.js パッケージマネージャーである を使用する方法について説明します CodeCatalyst。

> **Note**
>
> CodeCatalyst はnode v4.9.1、 以降および npm v5.0.0 以降をサポートしています。

トピック

npm の設定と使用

npm で を使用するには CodeCatalyst、パッケージリポジトリnpmに接続し、認証用の個人アクセストークン (PAT) を提供する必要があります。パッケージリポジトリnpmへの接続手順は、CodeCatalyst コンソールで確認できます。

目次

で npm を設定する CodeCatalyst

次の手順では、 CodeCatalyst パッケージリポジトリnpmを認証して接続する方法について説明します。npm の詳細については、npm の公式ドキュメント を参照してください。

CodeCatalyst パッケージリポジトリ**npm**に接続するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトに移動します。
3. ナビゲーションペインで、[Packages (パッケージ)] を選択します。
4. リストからパッケージリポジトリを選択します。
5. リポジトリに接続 を選択します。
6. 設定の詳細 のパッケージマネージャークライアント で、npm クライアント を選択します。
7. オペレーティングシステムを選択して、対応する設定ステップを表示します。
8. で npm を認証するには、個人用アクセストークン (PAT) が必要です CodeCatalyst。トークンが既にある場合は、それを使用できます。そうでない場合は、次のステップを使用して作成できます。

   a. (オプション): PAT 名 と有効期限 を更新します。

   b. トークンの作成 を選択します。

   c. PAT をコピーして安全な場所に保存します。

   > ⚠ Warning
   >
   > ダイアログボックスを閉じた後、PAT を再度表示またはコピーすることはできません。認証情報は、悪用後に攻撃者が認証情報を使用できる時間を最小限に抑えるために、有効期間を短くする必要があります。

9. プロジェクトのルートディレクトリから次のコマンドを実行して、パッケージリポジトリで npm を設定します。コマンドは以下を実行します。

   - プロジェクトに がない場合は、プロジェクトレベルの `.npmrc` ファイルを作成します。
   - パッケージリポジトリのエンドポイント情報をプロジェクトレベルの `.npmrc` ファイルに追加します。
   - 認証情報 (PAT) をユーザーレベルの `.npmrc` ファイルに追加します。

   次の値を置き換えます。

> **Note**
>
> コンソールの手順からコピーする場合、次のコマンドの値が更新され、変更する必要はありません。

- *username* を CodeCatalyst ユーザー名に置き換えます。
- *PAT* を PAT CodeCatalyst に置き換えます。
- *space_name* を CodeCatalyst スペース名に置き換えます。
- *proj_name* を CodeCatalyst プロジェクト名に置き換えます。
- *repo_name* を CodeCatalyst パッケージリポジトリ名に置き換えます。

```
npm set registry=https://packages.region.codecatalyst.aws/npm/space-name/proj-name/repo-name/ --location project
npm set //packages.region.codecatalyst.aws/npm/space-name/proj-name/repo-name/:_authToken=username:PAT
```

npm 6 以下の場合： npm が常に認証トークンを に渡すようにするには CodeCatalyst、GETリクエストに対しても、npm config set次のように always-auth 設定変数を に設定します。

```
npm set //packages.region.codecatalyst.aws/npm/space-name/proj-name/repo-name/:always-auth=true --location project
```

パッケージリポジトリからの npm CodeCatalyst パッケージのインストール

「」の手順に従って npm をリポジトリに接続すると[で npm を設定する CodeCatalyst](#)、リポジトリでnpmコマンドを実行できます。

npm install コマンドを使用して、パッケージリポジトリまたはそのアップストリームリポジトリの 1 つにある npm CodeCatalyst パッケージをインストールできます。

```
npm install lodash
```

から npmjs への npm パッケージのインストール CodeCatalyst

npm パッケージは、npmjs.com、 に接続されたゲートウェイ CodeCatalyst リポジトリへのアップストリーム接続を使用してリポジトリを設定することで、リポジトリを介して npmjs.com からインストールできますnpm-public-registry-gateway。npmjs からインストールされたパッケージは、ゲートウェイリポジトリと最も遠いダウンストリームパッケージリポジトリに取り込まれて保存されます。

npmjs からパッケージをインストールするには

1. まだ設定していない場合は、「」の手順に従ってパッケージリポジトリnpmで CodeCatalyst を設定しますで npm を設定する CodeCatalyst。
2. リポジトリがゲートウェイリポジトリ をアップストリーム接続npm-public-registry-gatewayとして追加していることを確認します。「」の手順に従ってnpm-public-registry-gatewayリポジトリアップストリームリポジトリの追加を選択すると、追加またはアップストリームソースnpm-public-registry-gatewayとして追加するアップストリームソースを確認できます。
3. `npm install` コマンドを使用してパッケージをインストールします。

```
npm install package_name
```

アップストリームリポジトリからのパッケージのリクエストの詳細については、「」を参照してくださいアップストリームリポジトリを持つパッケージバージョンのリクエスト。

パッケージリポジトリへの npm CodeCatalyst パッケージの発行

を完了したらで npm を設定する CodeCatalyst、npmコマンドを実行できます。

`npm publish` コマンドを使用して、npm パッケージを CodeCatalyst パッケージリポジトリに発行できます。

```
npm publish
```

npm パッケージを作成する方法については、「npm Docs での Node.js モジュールの作成」を参照してください。

npm コマンドサポート

以下のセクションでは、サポートされていない特定のnpmコマンドを一覧表示するだけでなく、パッケージリポジトリで CodeCatalystサポートされているコマンドを要約します。

トピック

パッケージリポジトリとやり取りするサポートされているコマンド

このセクションでは、npmクライアントが設定されたレジストリに対して 1 つ以上のリクエストを行うnpmコマンドを一覧表示します (例: `npm config set registry`)。これらのコマンドは、CodeCatalyst パッケージリポジトリに対して呼び出されたときに正しく機能することが検証されています。

コマンド	説明
bugs (バグ)	パッケージのバグトラッカー URL の場所を推測し、開こうとします。
ci (クリーンスレートインストール)	クリーンスレートでプロジェクトをインストールします。
deprecate (非推奨)	パッケージのバージョンを非推奨にします。
dist-tag (配布タグ)	パッケージ配布タグを変更します。
docs (ドキュメンテーション)	パッケージのドキュメント URL の場所を推測し、config パラメータを使用してその URL `--browser` を開くようにします。
doctor (ドクター)	一連のチェックを実行して、npm インストールが JavaScript パッケージを管理できることを検証します。
install (インストール)	パッケージをインストールします。
install-ci-test	クリーンスレートでプロジェクトをインストールし、テストを実行します。エイリアス: `npm cit` このコマンドは を実行し`npm ci`、その後すぐに を実行します`npm test`。

コマンド	説明
install-test (インストールテスト)	パッケージをインストールしてテストを実行します。を実行しnpm install、その後すぐにを実行しますnpm test。
outdated (旧式)	設定されたレジストリをチェックして、インストールされているパッケージが古くなっているかどうかを確認します。
ping (旧式)	設定または指定された npm レジストリに ping を実行し、認証を検証します。
publish (出力)	パッケージバージョンをレジストリに出力します。
update (アップデート)	パッケージのリポジトリ URL の場所を推測し、config パラメータを使用してそのリポジトリ URL --browser を開くようにします。
view (表示)	パッケージメタデータの表示 メタデータプロパティの印刷にも使用できます。

サポートされているクライアント側コマンド

これらのコマンドは、パッケージリポジトリと直接やり取りする必要がないため CodeCatalyst、それらをサポートするものは必要ありません。

コマンド	説明
bin (レガシー)	npm bin ディレクトリを表示します。
buid (構築)	パッケージを構築します。
cache (キャッシュ)	パッケージキャッシュを操作します。
completion (完成)	すべての npm コマンドでタブ補完を有効にします。

コマンド	説明
config (設定)	ユーザーとグローバル npmrc ファイルのコンテンツを更新します。
depute (代理)	ローカルパッケージツリーを検索し、依存関係をツリーの上位に移動して構造を簡素化しようとします。このツリーでは、複数の依存パッケージでより効果的に共有できます。
edit (編集)	インストールされたパッケージを編集します。現在の作業ディレクトリの依存関係を選択し、デフォルトの工ディタでパッケージディレクトリを開きます。
explore (調査)	インストールされているパッケージを参照します。指定されたインストール済みパッケージのディレクトリにサブシェルをスポーンします。コマンドが指定されている場合、そのコマンドはサブシェルで実行され、すぐにシャットダウンされます。
help (ヘルプ)	npm に関するヘルプを取得します。
help-search (ヘルプ検索)	npm ヘルプドキュメントを検索します。
init (初期)	package.json ファイルを作成します。
link (リンク)	パッケージディレクトリをシンボリックリンクします。
ls (リスト)	インストールされているパッケージを一覧表示します。
pack (パッケージ)	パッケージから tarball を作成します。
prefix (プレフィックス)	プレフィックスを表示します。も指定されていない限り、これはpackage.json ファイルを含む最も近い親ディレクトリ-gです。

コマンド	説明
prune (削除)	親パッケージの依存関係リストに一覧表示されていないパッケージを削除します。
rebuild (再構築)	一致したフォルダに対して npm build コマンドを実行します。
restart (再起動)	パッケージの停止、再起動、開始スクリプト、および関連するプリスクリプトとポストスクリプトを実行します。
root (ルート)	有効なnode_modules ディレクトリを標準出力に出力します。
run-script (スクリプト実行)	任意のパッケージスクリプトを実行します。
shrinkwrap (収縮包装)	パブリケーションの依存関係バージョンをロックダウンします。
uninstall (アンインストール)	パッケージをアンインストールします。

サポートされていないコマンド

これらのnpmコマンドは CodeCatalyst パッケージリポジトリではサポートされていません。

コマンド	説明	メモ
access (アクセス)	公開パッケージのアクセスレベルを設定します。	CodeCatalyst は、パブリック npmjs リポジトリとは異なるアクセス許可モデルを使用します。
adduser	レジストリユーザーアカウントを追加します。	CodeCatalyst は、パブリック npmjs リポジトリとは異なるユーザーモデルを使用します。

コマンド	説明	メモ
audit (監査)	セキュリティ監査を実行します。	CodeCatalyst は現在、セキュリティ脆弱性データを提供していません。
hook (フック)	追加、削除、リスト、更新など、npm フックを管理します。	CodeCatalyst は現在、変更通知メカニズムをサポートしていません。
login (ログイン)	ユーザーを認証します。これは `npm adduser` のエイリアスです。	CodeCatalyst は、パブリック npmjs リポジトリとは異なる認証モデルを使用します。詳細については、「で npm を設定する CodeCatalyst」を参照してください。
logout (サインアウト)	レジストリからサインアウトします。	CodeCatalyst は、パブリック npmjs リポジトリとは異なる認証モデルを使用します。CodeCatalyst リポジトリからサインアウトする方法はありませんが、認証トークンは設定可能な有効期限後に期限切れになります。デフォルトのトークンの期間は 12 時間です。
owner (オーナー)	パッケージの所有者を管理します。	CodeCatalyst は、パブリック npmjs リポジトリとは異なるアクセス許可モデルを使用します。
profile (プロファイル)	レジストリプロファイルの設定を変更します。	CodeCatalyst は、パブリック npmjs リポジトリとは異なるユーザーモデルを使用します。

コマンド	説明	メモ
search (検索)	検索語に一致するパッケージをレジストリで検索します。	CodeCatalyst は `search` コマンドをサポートしていません。
star (星)	お気に入りのパッケージをマークします。	CodeCatalyst は現在、お気に入りメカニズムをサポートしていません。
stars (星)	お気に入りとしてマークされたパッケージを表示します。	CodeCatalyst は現在、お気に入りメカニズムをサポートしていません。
team (チーム)	チームおよびチームメンバーシップを管理します。	CodeCatalyst は、パブリック npmjs リポジトリとは異なるユーザーおよびグループメンバーシップモデルを使用します。
token (トークン)	認証トークンを管理します。	CodeCatalyst は、認証トークンを取得するために別のモデルを使用します。詳細については、「で npm を設定する CodeCatalyst」を参照してください。
unpublished	レジストリからパッケージを削除します。	CodeCatalyst では、npm クライアントを使用したリポジトリからのパッケージバージョンの削除はサポートされていません。コンソールでパッケージを削除できます。
whoami (私は誰)	npm ユーザー名を表示します。	CodeCatalyst は、パブリック npmjs リポジトリとは異なるユーザーモデルを使用します。

npm タグ処理

npm レジストリは タグ をサポートしており、これはパッケージバージョンの文字列エイリアスです。バージョン番号を使用する代わりに、タグを使用してエイリアスを指定できます。例えば、複数の開発ストリームを持つプロジェクトがあり、ストリームごとに異なるタグ (`stable`、`beta`、`dev`、 など) を使用しているとしますcanary。詳細については、npm Docs の dist-tag を参照してください。

デフォルトでは、npm は `latest` タグを使用して、パッケージの現在のバージョンを識別します。`npm install` *`pkg`* (`@`*`version`* または `@`*`tag`* 指定子なし) は latest タグをインストールします。通常、プロジェクトは安定したリリースバージョンにのみ最新のタグを使用します。他のタグは、不安定版またはプレリリースバージョンに使用されます。

npm クライアントでのタグの編集

3 つの`npm dist-tag`コマンド (`add`、`rm`、および `ls`) は、デフォルトの npm レジストリ で機能するのと同じ方法で CodeCatalyst パッケージリポジトリで機能します。

npm タグと上流リポジトリ

がパッケージnpmのタグを要求し、そのパッケージのバージョンがアップストリームリポジトリにも存在する場合、 はタグを CodeCatalyst マージしてからクライアントに返します。例えば、 という名前のリポジトリには、 という名前のアップストリームリポジトリRがありますU。次の表は、両方のリポジトリに存在する `web-helper` という名前のパッケージのタグを示しています。

リポジトリ	パッケージ名	パッケージタグ
R	`web-helper`	latest (バージョン 1.0.0 のエイリアス)
U	`web-helper`	alpha (バージョン 1.0.1 のエイリアス)

この場合、npm クライアントがリポジトリ からweb-helperパッケージのタグを取得するとR、最新のタグとアルファタグの両方を受け取ります。タグが指すバージョンは変更されません。

アップストリームリポジトリとローカルリポジトリの両方の同じパッケージに同じタグが存在する場合、 は最後に更新されたタグ CodeCatalyst を使用します。例えば、ウェブヘルパー 上のタグが次のように変更したとします。

リポジトリ	パッケージ名	パッケージタグ	最終更新日
R	`web-helper`	latest (バージョン 1.0.0 のエイリアス)	2023 年 1 月 1 日
U	`web-helper`	latest (バージョン 1.0.1 のエイリアス)	2023 年 6 月 1 日

この場合、npm クライアントがリポジトリ からパッケージ web-helper のタグを取得するとR、最新のタグは最後に更新されたため、バージョン 1.0.1 にエイリアスを付けます。これにより、 を実行することで、ローカルリポジトリにまだ存在しないアップストリームリポジトリの新しいパッケージバージョンを簡単に使用できますnpm update。

パッケージのクォータ

次の表に、Amazon のパッケージのクォータと制限を示します CodeCatalyst。Amazon のクォータの詳細については CodeCatalyst、「」を参照してくださいのクォータ CodeCatalyst。

リソース	デフォルトのクォータ
パッケージリポジトリ	スペースあたり最大 1000 個。
直接アップストリームリポジトリ	パッケージリポジトリあたり最大 10 個。
アップストリームパッケージリポジトリが検索されました	リクエストされたパッケージバージョンごとに検索されるアップストリームリポジトリは最大 25 個です。
パッケージアセットファイルサイズ	パッケージアセットあたり最大 5GB。
パッケージアセット	パッケージバージョンあたり最大 150。

でワークフローを使用して構築、テスト、デプロイする CodeCatalyst

CodeCatalyst 開発環境でアプリケーションコードを記述し、CodeCatalyst ソースリポジトリ にプッシュしたら、デプロイする準備が整います。これを自動的に行う方法は、ワークフローを通じて行います。

ワークフローは、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) システムの一部としてコードを構築、テスト、デプロイする方法を説明する自動化された手順です。ワークフローは、ワークフローの実行中に実行する一連のステップまたはアクションを定義します。ワークフローは、ワークフローを開始するイベント、またはトリガー も定義します。ワークフローを設定するには、CodeCatalyst コンソールのビジュアルまたは YAML エディタ を使用してワークフロー定義ファイルを作成します。

> **Tip**
>
> プロジェクトでワークフローを使用する方法を簡単に確認するには、設計図 を使用してプロジェクトを作成します。各ブループリントは、レビュー、実行、および実験できる機能するワークフローをデプロイします。

ワークフロー定義ファイルについて

ワークフロー定義ファイルは、ワークフローを説明する YAML ファイルです。ファイルは、ソースリポジトリ のルートにある~/.codecatalyst/workflows/フォルダに保存されます。ファイルには、.yml または .yaml 拡張子を付けることができます。

以下は、シンプルなワークフロー定義ファイルの例です。この例の各行については、次の表で説明します。

```
Name: MyWorkflow
SchemaVersion: 1.0
RunMode: QUEUED
Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
```

```
  Build:
    Identifier: aws/build@v1
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      Steps:
        - Run: docker build -t MyApp:latest .
```

線グラフ	説明
`Name: MyWorkflow`	ワークフローの名前を指定します。Name プロパティの詳細については、「」を参照してください最上位のプロパティ。
`SchemaVersion: 1.0`	ワークフロースキーマのバージョンを指定します。SchemaVersion プロパティの詳細については、「」を参照してください最上位のプロパティ。
`RunMode: QUEUED`	が複数の実行 CodeCatalyst を処理する方法を示します。実行モードの詳細については、「」を参照してください実行のキューイング動作の設定。
`Triggers:`	ワークフロー実行を開始するロジックを指定します。トリガーについての詳細は、「トリガーを使用したワークフローの自動実行の開始」を参照してください。

`- Type: PUSH`
`  Branches:`
`  - main`
	デフォルトのソースリポジトリのmainブランチにコードをプッシュするたびにワークフローを開始する必要があることを示します。ワークフローソースの詳細については、「」を参照してくださいワークフローをソースリポジトリに接続する。

`Actions:`	ワークフローの実行中に実行するタスクを定義します。この例では、 Actionsセクションで

線グラフ	説明
	という 1 つのアクションを定義しますBuild。アクションの詳細については、「」を参照してくださいワークフローが実行するアクションの設定。
`Build:`	Build アクションのプロパティを定義します。ビルドアクションの詳細については、「」を参照してくださいワークフローによる構築。
`Identifier: aws/build@v1`	ビルドアクションの一意のハードコードされた識別子を指定します。

`Inputs:`
`  Sources:`
`  - WorkflowSource`
	ビルドアクションがWorkflowSource ソースリポジトリを検索して、処理を完了するために必要なファイルを見つける必要があることを示します。詳細については、「ワークフローをソースリポジトリに接続する」を参照してください。

`Configuration:`	ビルドアクションに固有の設定プロパティが含まれます。

`Steps:`
`  - Run: docker build -t MyApp:lat`
`est .`
	という名前の Docker イメージをビルドMyAppし、 でタグ付けするようにビルドアクションに指示しますlatest。

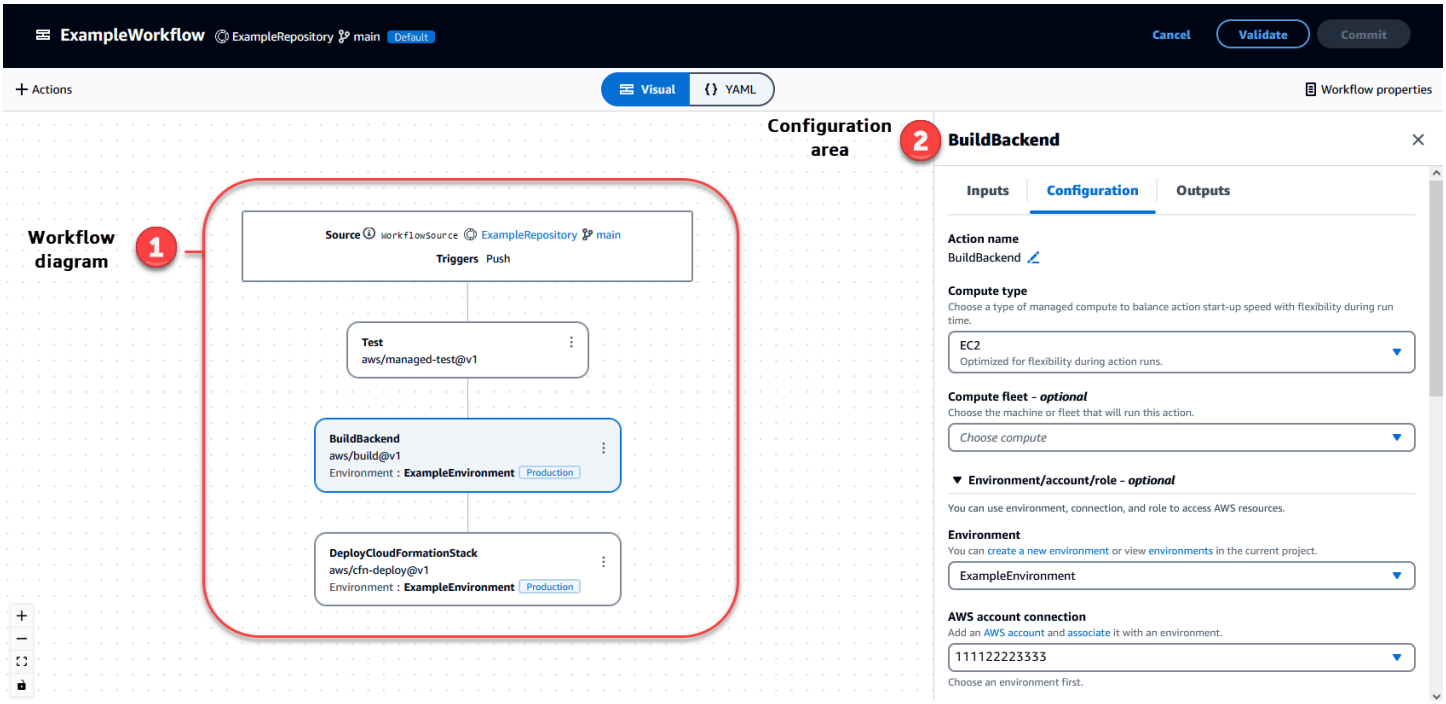
ワークフロー定義ファイルで使用できるすべてのプロパティの完全なリストについては、「」を参照してくださいワークフロー YAML 定義。

CodeCatalyst コンソールのビジュアルエディタと YAML エディタの使用

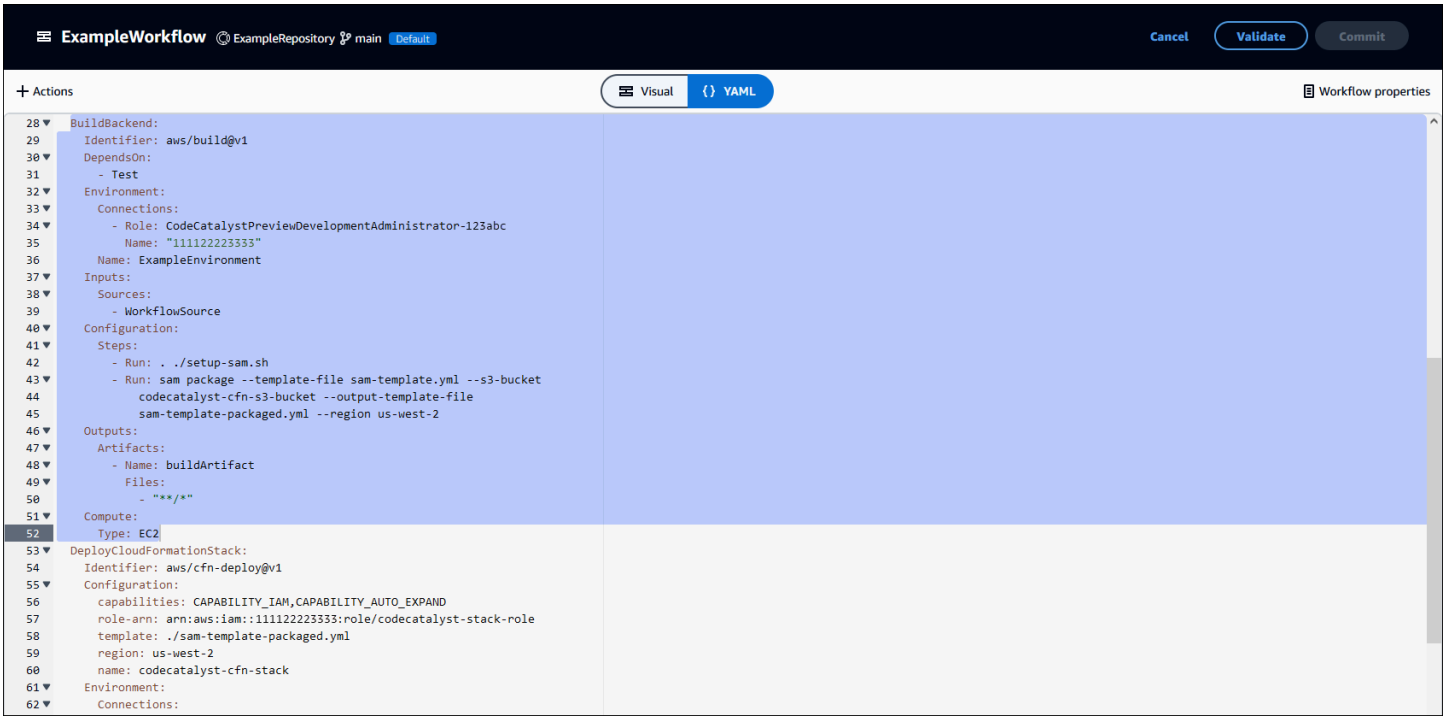
ワークフロー定義ファイルを作成および編集するには、任意のエディタを使用できますが、CodeCatalyst コンソールのビジュアルエディタまたは YAML エディタを使用することをお勧めしま

す。これらのエディタは、YAML プロパティ名、値、ネスト、間隔、大文字などが正しいことを確認するのに役立つファイル検証を提供します。

次の図は、ビジュアルエディタのワークフローを示しています。ビジュアルエディタには、ワークフロー定義ファイルを作成および設定するための完全なユーザーインターフェイスが用意されています。ビジュアルエディタには、ワークフローの主要コンポーネントを示すワークフロー図 (1) と設定領域 (2) が含まれています。



または、次の図に示す YAML エディタを使用することもできます。YAML エディタを使用して、大きなコードブロック (チュートリアルなど) に貼り付けたり、ビジュアルエディタでは提供されない高度なプロパティを追加したりできます。

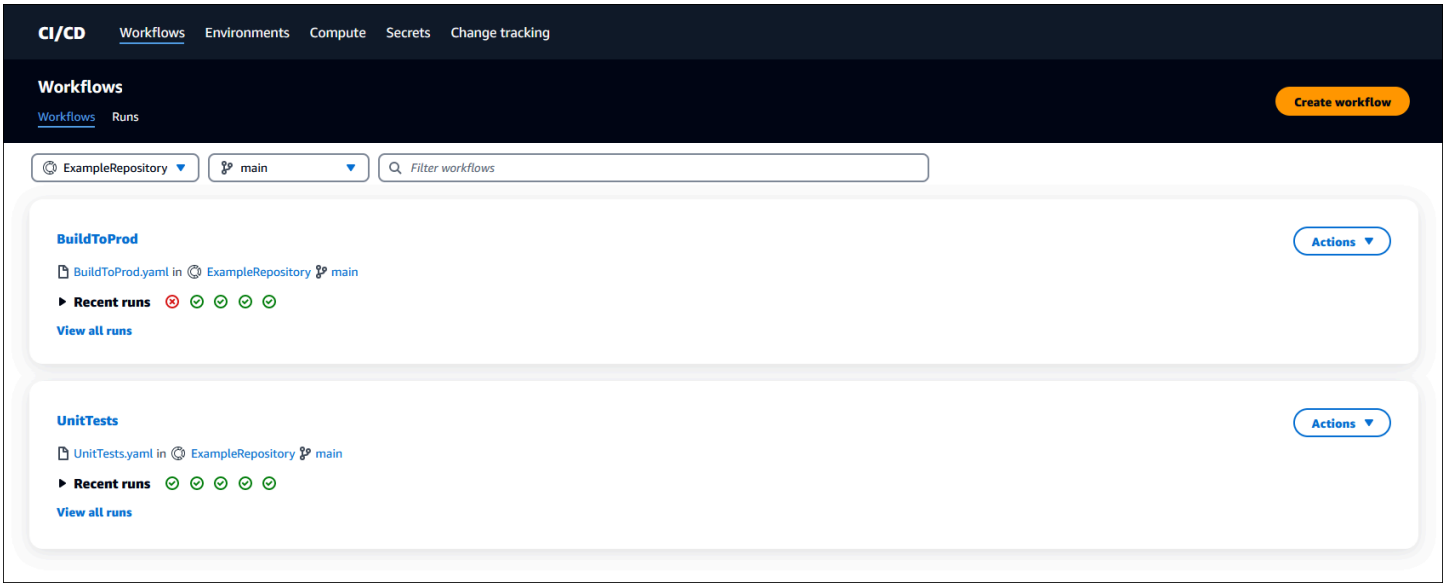


ビジュアルエディタから YAML エディタに切り替えて、設定が基盤となる YAML コードに与える影響を確認できます。

ワークフローの検出

ワークフローの概要ページと、同じプロジェクトで設定した他のワークフローでワークフローを表示できます。

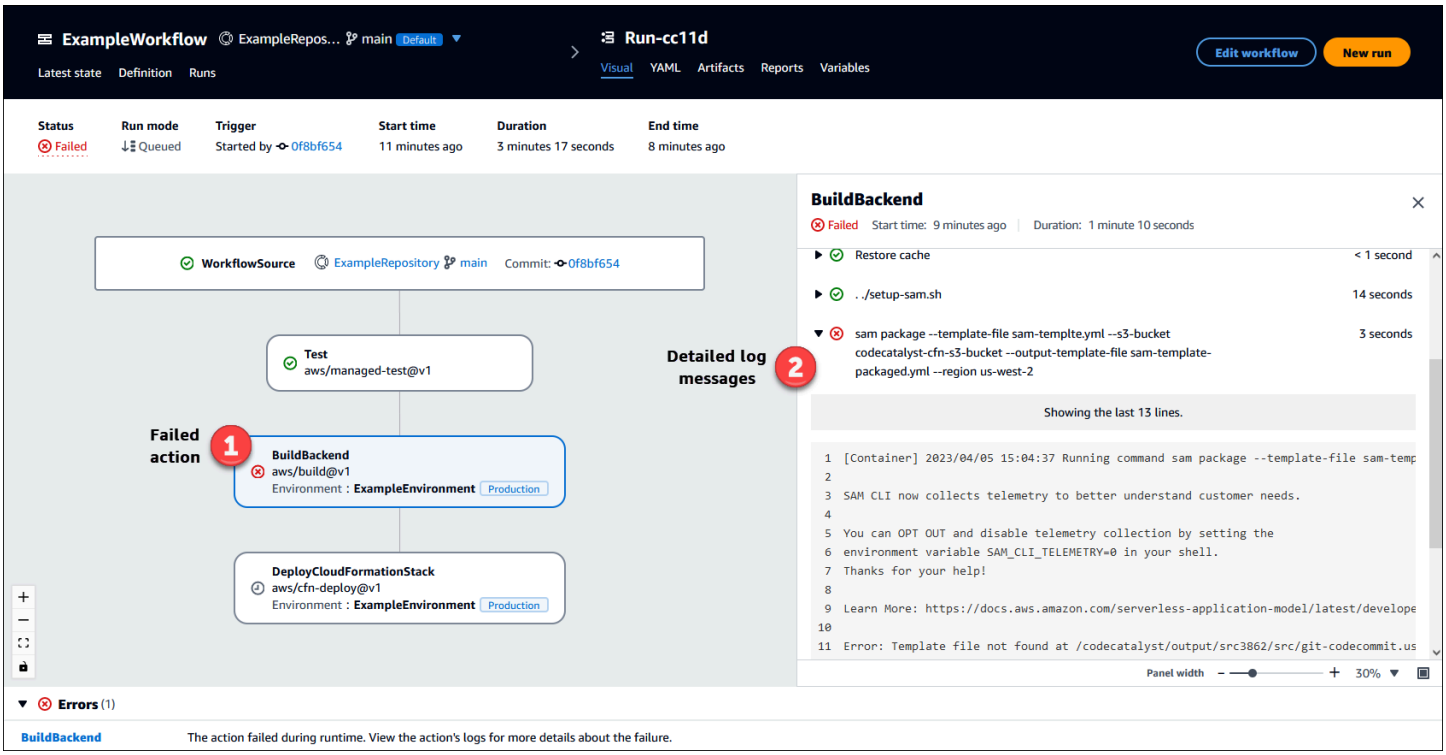
次の図は、ワークフローの概要ページを示しています。と の 2 BuildToProd つのワークフローが入力されますUnitTests。両方が数回実行されたことがわかります。最近の実行を選択して実行履歴をすばやく表示するか、ワークフローの名前を選択してワークフローの YAML コードやその他の詳細情報を表示できます。



ワークフロー実行の詳細の表示

ワークフローの概要ページで実行を選択すると、ワークフロー実行の詳細を表示できます。

次の図は、ソースへのコミット時に自動的に開始された Run-cc11d というワークフロー実行の詳細を示しています。ワークフロー図は、アクションが失敗したことを示しています (1)。ログ (2) に移動して、詳細なログメッセージを表示し、問題をトラブルシューティングできます。ワークフロー実行の詳細については、「」を参照してください[ワークフローの実行](#)。



次のステップ

ワークフローの概念の詳細については、「」を参照してくださいワークフローの概念。

最初のワークフローを作成するには、「」を参照してくださいワークフローの開始方法。

ワークフローの概念

でワークフローを使用してコードを構築、テスト、またはデプロイするときに知っておくべき概念と用語をいくつか紹介します CodeCatalyst。

ワークフロー

ワークフローは、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) システムの一部としてコードを構築、テスト、デプロイする方法を説明する自動化された手順です。ワークフローは、ワークフローの実行中に実行する一連のステップまたはアクションを定義します。ワークフローは、ワークフローを開始するイベント、またはトリガー も定義します。ワークフローを設定するには、CodeCatalyst コンソールのビジュアルまたは YAML エディタ を使用してワークフロー定義ファイルを作成します。

> **Tip**
>
> プロジェクトでワークフローを使用する方法を簡単に確認するには、設計図 を使用してプロジェクトを作成します。各ブループリントは、レビュー、実行、および実験できる機能するワークフローをデプロイします。

ワークフロー定義ファイル

ワークフロー定義ファイルは、ワークフローを記述する YAML ファイルです。ファイルは、ソースリポジトリ のルートにある`~/.codecatalyst/workflows/`フォルダに保存されます。ファイルには、.yml または .yaml 拡張子を付けることができます。

ワークフロー定義ファイルの詳細については、「」を参照してくださいワークフロー YAML 定義。

アクション

アクションはワークフローの主要な構成要素であり、ワークフローの実行中に実行する作業またはタスクの論理単位を定義します。通常、ワークフローには、設定方法に応じて順番または並行して実行される複数のアクションが含まれます。

アクションの詳細については、「」を参照してくださいワークフローが実行するアクションの設定。

アクショングループ

アクショングループには、1 つ以上のアクションが含まれます。アクションをアクショングループにグループ化すると、ワークフローを整理しやすくなり、異なるグループ間の依存関係を設定することもできます。

アクショングループの詳細については、「」を参照してくださいアクションをアクショングループにグループ化する。

アーティファクト

アーティファクトはワークフローアクションの出力であり、通常はファイルのフォルダまたはアーカイブで構成されます。アーティファクトは、ファイルや情報をアクション間で共有できるため、重要です。

アーティファクトの詳細については、「アーティファクトを使用したワークフロー内のアクション間のデータの共有」を参照してください。

コンピューティング

コンピューティングとは、ワークフローアクションを実行 CodeCatalyst するために が管理および保守するコンピューティングエンジン (CPU、メモリ、オペレーティングシステム) を指します。

コンピューティングの詳細については、「」を参照してくださいワークフローのコンピューティング環境とランタイム環境の Docker イメージの設定。

環境

開発環境 と混同されない環境 は、コードがデプロイされる場所です。通常、実行中のアプリケーションのインスタンスと関連するインフラストラクチャが含まれます。開発、テスト、ステージ

ング、本番稼働などの名前を環境に付けます。によって生成された環境 CodeCatalyst へのデプロイは、環境ページに表示されます。環境を設定するには、 などの名前を付けmy-production-environment、それを に関連付けます AWS アカウント。

環境は、デプロイ情報の表示に加えて、ワークフローアクション に AWS IAM ロールを割り当てるメカニズムとしても機能します。

環境の詳細については、「」を参照してください環境を使用して AWS アカウント および VPCsにデプロイする CodeCatalyst 。

ゲート

ゲートは、特定の条件が満たされない限り、ワークフローの実行が続行されないようにするために使用できるワークフローコンポーネントです。ゲートの例は承認ゲートです。承認ゲートでは、ワークフローの実行を続行する前にコンソール CodeCatalystで承認を送信する必要があります。

ワークフロー内の一連のアクション間、または最初のアクションの前 (ソースのダウンロードの直後に実行される) にゲートを追加できます。最後のアクションの後にゲートを追加する必要がある場合は、追加することもできます。

ゲートの詳細については、「」を参照してくださいワークフロー実行のゲート設定。

ゲート

ゲートは、特定の条件が満たされない限り、ワークフローの実行が続行されないようにするために使用できるワークフローコンポーネントです。ゲートの例は承認ゲートです。承認ゲートでは、ワークフローの実行を続行する前にコンソール CodeCatalystで承認を送信する必要があります。

ワークフロー内の一連のアクション間にゲートを追加するか、最初のアクションの前にゲートを追加できます (ソースのダウンロードの直後に実行されます) 。最後のアクションの後にゲートを追加する必要がある場合は、追加することもできます。

ゲートの詳細については、「」を参照してくださいワークフロー実行のゲート設定。

レポート

レポートには、ワークフローの実行中に発生するテストの詳細が含まれます。テストレポート、コードカバレッジレポート、ソフトウェア構成分析レポート、静的分析レポートなどのレポートを作成できます。レポートを使用すると、ワークフロー中の問題のトラブルシューティングに役立ちます。複

数のワークフローからのレポートが多数ある場合は、レポートを使用して傾向と障害率を表示し、アプリケーションとデプロイ設定の最適化に役立てることができます。

レポートの詳細については、「品質レポートタイプ」を参照してください。

実行

実行はワークフローの 1 回の反復です。実行中、 CodeCatalyst はワークフロー設定ファイルで定義されたアクションを実行し、関連するログ、アーティファクト、変数を出力します。

実行の詳細については、「」を参照してくださいワークフローの実行。

[Sources] (出典)

ソース は、入力ソース とも呼ばれ、オペレーションの実行に必要なファイルを取得するためにワークフローアクションが接続するソースリポジトリです。例えば、ワークフローアクションは、ソースリポジトリに接続して、アプリケーションを構築するためにアプリケーションソースファイルを取得する場合があります。

sources の詳細については、「ワークフローをソースリポジトリに接続する」を参照してください。

変数

変数は、 CodeCatalyst ワークフローで参照できる情報を含むキーと値のペアです。

変数の詳細については、「」を参照してくださいワークフローでの変数の設定と使用。

ワークフロートリガー

ワークフロートリガー 、または単にトリガー を使用すると、コードプッシュなどの特定のイベントが発生したときに、ワークフローの実行を自動的に開始できます。トリガーを設定して、ソフトウェアデベロッパーが CodeCatalyst コンソールからワークフロー実行を手動で開始する必要がないようにすることもできます。

次の 3 種類のトリガーを使用できます。

- プッシュ — コードプッシュトリガーにより、コミットがプッシュされるたびにワークフロー実行が開始されます。
- プルリクエスト – プルリクエストトリガーは、プルリクエストが作成、改訂、またはクローズされるたびにワークフロー実行を開始します。

- スケジュール – スケジュールトリガーにより、定義したスケジュールでワークフロー実行が開始されます。スケジュールトリガーを使用してソフトウェアの夜間ビルドを実行し、ソフトウェアデベロッパーが翌日に作業できるようにすることを検討してください。

プッシュ、プルリクエスト、スケジュールトリガーは、単独で使用することも、同じワークフロー内で組み合わせて使用することもできます。

トリガーについての詳細は、「トリガーを使用したワークフローの自動実行の開始」を参照してください。

ワークフローの開始方法

このチュートリアルでは、最初のワークフローを作成して設定する方法について説明します。

> **Tip**
>
> 事前設定済みのワークフローから始めることをお勧めしますか？「」を参照してください。これには設計図を使用したプロジェクトの作成、機能するワークフロー、サンプルアプリケーション、およびその他のリソースを使用してプロジェクトをセットアップする手順が含まれています。

トピック

前提条件

開始する前に:

- CodeCatalyst スペース が必要です。詳細については、「スペースの作成」を参照してください。
- CodeCatalyst スペースには、空の Start from scratch CodeCatalyst プロジェクトが必要です。

```
codecatalyst-project
```

詳細については、「Amazon での空のプロジェクトの作成 CodeCatalyst」を参照してください。

- プロジェクトには、次の CodeCatalyst リポジトリが必要です。

```
codecatalyst-source-repository
```

詳細については、「ソースリポジトリの作成」を参照してください。

> **Note**
>
> 既存のプロジェクトとソースリポジトリがある場合は、それらを使用できますが、新しいプロジェクトを作成すると、このチュートリアルの最後にクリーンアップが簡単になります。

ステップ 1: ワークフローを作成して設定する

このステップでは、変更が行われたときにソースコードを自動的に構築してテストするワークフローを作成して設定します。

ワークフローを作成するには

1. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
2. ワークフローの作成 を選択します。

   ワークフロー定義ファイルは、 CodeCatalyst コンソールの YAML エディタに表示されます。

ワークフローを設定するには

ワークフローは、ビジュアルエディタまたは YAML エディタで設定できます。YAML エディタから始めて、ビジュアルエディタに切り替えてみましょう。

1. + Actions を選択すると、ワークフローに追加できるワークフローアクションのリストが表示されます。
2. ビルドアクションで + を選択して、アクションの YAML をワークフロー定義ファイルに追加します。ワークフローは次のようになります。

```
Name: Workflow_fe47
SchemaVersion: "1.0"

# Optional - Set automatic triggers.
Triggers:
  - Type: Push
    Branches:
      - main

# Required - Define action configurations.
Actions:
  Build_f0:
    Identifier: aws/build@v1

    Inputs:
      Sources:
        - WorkflowSource # This specifies that the action requires this workflow as
 a source

    Outputs:
      AutoDiscoverReports:
        Enabled: true
        # Use as prefix for the report files
        ReportNamePrefix: rpt

    Configuration:
      Steps:
        - Run: echo "Hello, World!"
        - Run: echo "<?xml version=\"1.0\" encoding=\"UTF-8\" ?>" >> report.xml
        - Run: echo "<testsuite tests=\"1\" name=\"TestAgentJunit\" >" >>
 report.xml
        - Run: echo "<testcase classname=\"TestAgentJunit\" name=\"Dummy
            Test\"/></testsuite>" >> report.xml
```

ワークフローは、WorkflowSourceソースリポジトリ内のファイルを Build_f0アクションを実行しているコンピューティングマシンにコピーし、ログHello, World!に出力し、コンピューティングマシンでテストレポートを検出して、 CodeCatalyst コンソールのレポートページに出力します。

3. ビジュアル を選択して、ビジュアルエディタでワークフロー定義ファイルを表示します。ビジュアルエディタのフィールドでは、YAML エディタに表示される YAML プロパティを設定できます。

ステップ 2: コミットを使用してワークフローを保存する

このステップでは、変更を保存します。ワークフローはリポジトリに.yamlファイルとして保存されるため、変更をコミットで保存します。

ワークフローの変更をコミットするには

1. (オプション) 検証 を選択して、ワークフローの YAML コードが有効であることを確認します。
2. [Commit] (コミット) を選択します。
3. ワークフローファイル名 に、 のようなワークフロー設定ファイルの名前を入力します**my-first-workflow**。
4. コミットメッセージ に、 のようなコミットを識別するメッセージを入力します**create my-first-workflow.yaml**。
5. リポジトリ で、ワークフローを () に保存するリポジトリを選択しますcodecatalyst-repository。
6. ブランチ名 で、ワークフローを () に保存するブランチを選択しますmain。
7. [Commit] (コミット) を選択します。

新しいワークフローがワークフローのリストに表示されます。表示されるまでに少し時間がかかる場合があります。

ワークフローはコミットとともに保存され、ワークフローにはコードプッシュトリガーが設定されているため、ワークフローを保存するとワークフローの実行が自動的に開始されます。

ステップ 3: 実行結果を表示する

このステップでは、コミットから開始された実行に移動し、結果を表示します。

実行結果を表示するには

1. ワークフローの名前を選択しますWorkflow_fe47。例えば、。

   ソースリポジトリのラベル (WorkflowSource) とビルドアクション (Build_f0 など) を示すワークフロー図。
2. ワークフロー実行図で、ビルドアクション (Build_f0 など) を選択します。

3. ログ 、レポート 、設定 、および変数 タブの内容を確認します。 これらのタブには、ビルドアクションの結果が表示されます。

   詳細については、「ビルドアクションの結果の表示」を参照してください。

（オプション）ステップ 4: クリーンアップする

このステップでは、このチュートリアルで作成したリソースをクリーンアップします。

リソースを削除するには

- このチュートリアル用に新しいプロジェクトを作成した場合は、削除します。手順については、「プロジェクトの削除」を参照してください。プロジェクトを削除すると、ソースリポジトリとワークフローも削除されます。

ワークフローによる構築

CodeCatalyst ワークフロー を使用すると、アプリケーションやその他のリソースを構築できます。

トピック

- アプリケーションを構築するにはどうすればよいですか？
- ビルドアクションの利点
- ビルドアクションの代替方法
- ビルドアクションの追加
- ビルドアクションの結果の表示
- チュートリアル: Amazon S3 にアーティファクトをアップロードする
- ビルドおよびテストアクションの YAML 定義

アプリケーションを構築するにはどうすればよいですか？

でアプリケーションまたはリソースを構築するには CodeCatalyst、まずワークフローを作成し、その中にビルドアクションを指定します。

ビルドアクションは、ソースコードをコンパイルし、ユニットテストを実行し、デプロイ可能なアーティファクトを生成するワークフロー構成要素です。

CodeCatalyst コンソールのビジュアルエディタまたは YAML エディタを使用して、ワークフローにビルドアクションを追加します。

アプリケーションまたはリソースを構築するための大まかな手順は次のとおりです。

アプリケーションを構築するには (高レベルタスク)

1. で CodeCatalyst、構築するアプリケーションのソースコードを追加します。詳細については、「のプロジェクトのリポジトリにソースコードを保存する CodeCatalyst」を参照してください。
2. で CodeCatalyst、ワークフロー を作成します。ワークフローでは、アプリケーションを構築、テスト、デプロイする方法を定義します。詳細については、「ワークフローの開始方法」を参照してください。
3. (オプション) ワークフローで、ワークフローを自動的に開始するイベントを示すトリガーを追加します。詳細については、「トリガーを使用したワークフローの自動実行の開始」を参照してください。
4. ワークフローでは、アプリケーションまたはリソースソースコードをコンパイルしてパッケージ化するビルドアクションを追加します。オプションで、これらの目的でテストまたはデプロイアクションを使用しない場合は、ビルドアクションでユニットテストを実行し、レポートを生成し、アプリケーションをデプロイすることもできます。テストおよびデプロイアクションの詳細については、「」を参照してくださいビルドアクションの追加。
5. (オプション) ワークフローで、テストアクションとデプロイアクションを追加して、アプリケーションまたはリソースをテストおよびデプロイします。Amazon ECS などのさまざまなターゲットにアプリケーションをデプロイするには、事前設定されたアクションから選択できます。詳細については、「ワークフローを使用したテスト」および「ワークフローを使用したデプロイ」を参照してください。
6. ワークフローは、手動またはトリガーを使用して自動的に開始します。ワークフローは、ビルド、テスト、デプロイのアクションを順番に実行して、アプリケーションとリソースをビルド、テスト、ターゲットにデプロイします。詳細については、「ワークフローの手動実行の開始」を参照してください。

ビルドアクションの利点

ワークフロー内でビルドアクションを使用すると、次の利点があります。

- フルマネージド – ビルドアクションにより、独自のビルドサーバーをセットアップ、パッチ適用、更新、管理する必要がなくなります。

- オンデマンド – ビルドアクションは、ビルドのニーズに合わせてオンデマンドでスケーリングされます。料金は、使用したビルド分数に対してのみ発生します。詳細については、「ワークフローのコンピューティング環境とランタイム環境の Docker イメージの設定」を参照してください。
- すぐに使用可能 — ビルドアクションを含むすべてのワークフローアクションの実行に使用される、事前にパッケージ化されたランタイム環境の Docker イメージ CodeCatalyst が含まれます。これらのイメージには、 AWS CLI や Node.js などのアプリケーションを構築するための便利なツールが事前設定されています。パブリックレジストリまたはプライベートレジストリから指定したビルドイメージを使用する CodeCatalyst ように を設定できます。詳細については、「ランタイム環境の Docker イメージの指定」を参照してください。

ビルドアクションの代替方法

ビルドアクションを使用してアプリケーションをデプロイする場合は、代わりにデプロイアクションの使用 CodeCatalystを検討してください。デプロイアクションは、ビルドアクションを使用している場合に手動で書き込む必要がある設定を実行します behind-the-scenes。使用可能なデプロイアクションの詳細については、「」を参照してくださいデプロイアクションのリスト。

AWS CodeBuild を使用してアプリケーションを構築することもできます。詳細については、「 CodeBuildとは」を参照してください。

ビルドアクションの追加

ワークフローにビルドアクションを追加するには、次の手順に従います CodeCatalyst 。

Visual

ビジュアルエディタを使用してビルドアクションを追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
3. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
4. [編集] を選択します。
5. ビジュアル を選択します。
6. [アクション] を選択します。

7. アクション で、ビルド を選択します。
8. 入力タブと設定タブで、必要に応じてフィールドに入力します。各フィールドの説明については、「」を参照してくださいビルドおよびテストアクションの YAML 定義。このリファレンスでは、YAML エディタとビジュアルエディタの両方に表示される各フィールド (および対応する YAML プロパティ値) に関する詳細情報を提供します。
9. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
10. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

YAML

YAML エディタを使用してビルドアクションを追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
3. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
4. [編集] を選択します。
5. YAML を選択します。
6. [アクション] を選択します。
7. アクション で、ビルド を選択します。
8. 必要に応じて YAML コードのプロパティを変更します。使用可能な各プロパティの説明は、「」に記載されていますビルドおよびテストアクションの YAML 定義。
9. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
10. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

ビルドアクション定義

ビルドアクションは、ワークフロー定義ファイル内の一連の YAML プロパティとして定義されます。これらのプロパティの詳細については、ビルドおよびテストアクションの YAML 定義「」の「」を参照してくださいワークフロー YAML 定義。

ビルドアクションの結果の表示

生成されたログ、レポート、変数など、ビルドアクションの結果を表示するには、以下の手順に従います。

ビルドアクションの結果を表示するには

1. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
2. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
3. ワークフロー図で、ビルドアクションの名前を選択します。例えば、ビルド です。
4. ビルド実行のログを表示するには、ログ を選択します。さまざまなビルドフェーズのログが表示されます。必要に応じて、ログを展開または折りたたむことができます。
5. ビルドアクションによって生成されたテストレポートを表示するには、レポート を選択するか、ナビゲーションペインでレポート を選択します。詳細については、「品質レポートタイプ」を参照してください。
6. ビルドアクションに使用される設定を表示するには、設定 を選択します。詳細については、「ビルドアクションの追加」を参照してください。
7. ビルドアクションで使用される変数を表示するには、変数 を選択します。詳細については、「ワークフローでの変数の設定と使用」を参照してください。

チュートリアル: Amazon S3 にアーティファクトをアップロードする

このチュートリアルでは、いくつかのビルドアクション を含む CodeCatalyst ワークフローを使用して Amazon S3 バケットにアーティファクトをアップロードする方法について説明します。これらのアクションは、ワークフローの開始時に連続して実行されます。最初のビルドアクションは、`Hello.txt`と の 2 つのファイルを生成し`Goodbye.txt`、それらをビルドアーティファクトにバンドルします。2 番目のビルドアクションは、アーティファクトを Amazon S3 にアップロードします。コミットをソースリポジトリにプッシュするたびに実行するようにワークフローを設定します。

トピック

- 前提条件
- ステップ 1: AWS ロールを作成する
- ステップ 2: Amazon S3 バケットを作成する
- ステップ 3: ソースリポジトリを作成する

- ステップ 4: ワークフローを作成する
- ステップ 5: 結果を確認する
- クリーンアップ

前提条件

開始するには、以下が必要です。

- 接続された AWS アカウントを持つ CodeCatalyst スペースが必要です。詳細については、「スペースの作成」を参照してください。
- スペースには、次のような空の「最初から開始 CodeCatalyst」プロジェクトが必要です。

```
codecatalyst-artifact-project
```

  詳細については、「Amazon での空のプロジェクトの作成 CodeCatalyst」を参照してください。
- プロジェクトには、次の CodeCatalyst 環境が必要です。

```
codecatalyst-artifact-environment
```

  この環境を次のように設定します。
  - 開発 など、任意のタイプを選択します。
  - AWS アカウントを接続します。

  詳細については、「環境を使用して AWS アカウント および VPCsにデプロイする CodeCatalyst」を参照してください。

ステップ 1: AWS ロールを作成する

このステップでは、後でワークフローのビルドアクションに割り当てる AWS IAM ロールを作成します。このロールは、ビルド CodeCatalyst アクションに AWS アカウントにアクセスし、アーティファクトが保存される Amazon S3 に書き込むアクセス許可を付与します。ロールはビルドロール と呼ばれます。

> **Note**
>
> 別のチュートリアル用に作成したビルドロールが既にある場合は、このチュートリアルでも使用できます。以下の手順で示されているアクセス許可と信頼ポリシーがあることを確認してください。

IAM ロールの詳細については、「 ユーザーガイド」の「IAM ロールAWS AWS Identity and Access Management 」を参照してください。

ビルドロールを作成するには

1. 次のように、ロールのポリシーを作成します。
    a. にサインインします AWS。
    b. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
    c. ナビゲーションペインで、ポリシー を選択します。
    d. ポリシーの作成を選択します。
    e. [JSON] タブを選択します。
    f. 既存のコードを削除します。
    g. 次のコードを貼り付けます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:ListBucket"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> ロールを初めて使用してワークフローアクションを実行するときは、リソースポリシーステートメントでワイルドカードを使用し、使用可能になった後にリソース名でポリシーの範囲を絞り込みます。
>
> ```
> "Resource": "*"
> ```

h. [次へ : タグ] を選択します。

i. [次へ: レビュー] を選択します。

j. 名前 で、次のように入力します。

```
codecatalyst-s3-build-policy
```

k. [ポリシーの作成] を選択します。

これで、アクセス許可ポリシーが作成されました。

2. 次のようにビルドロールを作成します。

a. ナビゲーションペインで ロール を選択してから、ロールを作成する を選択します。

b. カスタム信頼ポリシー を選択します。

c. 既存のカスタム信頼ポリシーを削除します。

d. 次のカスタム信頼ポリシーを追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
```

```
    ]
}
```

e. [次へ] をクリックします。

f. アクセス許可ポリシー で、そのチェックボックスを検索codecatalyst-s3-build-policyして選択します。

g. [次へ] をクリックします。

h. ロール名 には、次のように入力します。

```
codecatalyst-s3-build-role
```

i. ロールの説明 には、次のように入力します。

```
CodeCatalyst build role
```

j. [ロールを作成] を選択します。

これで、信頼ポリシーとアクセス許可ポリシーを使用してビルドロールが作成されました。

ステップ 2: Amazon S3 バケットを作成する

このステップでは、 Hello.txtおよび Goodbye.txtアーティファクトをアップロードする Amazon S3 バケットを作成します。

Amazon S3 バケットを作成するには

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. メインペインで、バケットの作成 を選択します。
3. バケット名 には、次のように入力します。

```
codecatalyst-artifact-bucket
```

4. [AWS リージョン] で、リージョンを選択します。このチュートリアルでは、米国西部 (オレゴン) us-west-2 を選択したことを前提としています。Amazon S3 でサポートされているリージョンの詳細については、「」の「Amazon Simple Storage Service エンドポイントとクォータ」を参照してくださいAWS 全般のリファレンス。
5. ページの下部で、バケットの作成 を選択します。
6. 先ほど作成したバケットの名前をコピーします。例：

```
codecatalyst-artifact-bucket
```

これで、米国西部 (オレゴン) us-west-2 リージョン**codecatalyst-artifact-bucket**に という名前のバケットが作成されました。

ステップ 3: ソースリポジトリを作成する

このステップでは、 でソースリポジトリを作成します CodeCatalyst。このリポジトリは、チュートリアルのワークフロー定義ファイルを保存するために使用されます。

ソースリポジトリの詳細については、「」を参照してくださいソースリポジトリの作成。

ソースリポジトリを作成するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクト に移動しますcodecatalyst-artifact-project。
3. ナビゲーションペインで [コード] を選択してから、[ソースリポジトリ] を選択します。
4. [リポジトリの追加] を選択し、[リポジトリの作成] を選択します。
5. リポジトリ名 で、次のように入力します。

```
codecatalyst-artifact-source-repository
```

6. [作成] を選択します。

これで、 というリポジトリが作成されましたcodecatalyst-artifact-source-repository。

ステップ 4: ワークフローを作成する

このステップでは、順番に実行される以下の構成要素で構成されるワークフローを作成します。

- トリガー — このトリガーは、変更をソースリポジトリにプッシュすると、ワークフローの実行を自動的に開始します。トリガーの詳細については、「」を参照してくださいトリガーを使用したワークフローの自動実行の開始。
- というビルドアクション GenerateFiles – トリガー時に、 GenerateFilesアクションは Hello.txtと の 2 つのファイルを作成しGoodbye.txt、 という出力アーティファクトにパッケージ化しますcodecatalystArtifact。

- という別のビルドアクション Upload – GenerateFilesアクションが完了すると、 Upload アクションは AWS CLI コマンドaws s3 syncを実行して、 codecatalystArtifactおよびソースリポジトリのファイルを Amazon S3 バケットにアップロードします。 AWS CLI はコンピューティングプラットフォームに CodeCatalystプリインストールおよび事前設定されているため、インストールまたは設定する必要はありません。

  CodeCatalyst コンピューティングプラットフォームでパッケージ化されたソフトウェアの詳細については、「」を参照してくださいランタイム環境の Docker イメージの指定。の aws s3 sync コマンドの詳細については、「 コマンドAWS CLI リファレンス AWS CLI」の「Sync」を参照してください。

ビルドアクションの詳細については、「」を参照してくださいワークフローによる構築。

ワークフローを作成するには

1. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
2. ワークフローの作成 を選択します。
3. YAML サンプルコードを削除します。
4. 次の YAML コードを追加します。

```
Name: codecatalyst-artifact-workflow
SchemaVersion: 1.0

Triggers:
  - Type: Push
    Branches:
      - main
Actions:
  GenerateFiles:
    Identifier: aws/build@v1
    Configuration:
      Steps:
        # Create the output files.
        - Run: echo "Hello, World!" > "Hello.txt"
        - Run: echo "Goodbye!" > "Goodbye.txt"
    Outputs:
      Artifacts:
        - Name: codecatalystArtifact
          Files:
            - "**/*"
```

```
  Upload:
    Identifier: aws/build@v1
    DependsOn:
      - GenerateFiles
    Environment:
      Name: codecatalyst-artifact-environment
      Connections:
        - Name: codecatalyst-account-connection
          Role: codecatalyst-s3-build-role
    Inputs:
      Artifacts:
        - codecatalystArtifact
    Configuration:
      Steps:
        # Upload the output artifact to the S3 bucket.
        - Run: aws s3 sync . s3://codecatalyst-artifact-bucket
```

上記のコードで、以下を置き換えます。

- *codecatalyst-artifact-environment* で作成した環境の名前前提条件。
- *codecatalyst-account-connection* で作成したアカウント接続の名前前提条件。
- *codecatalyst-s3-build-role* で作成したビルドロールの名前ステップ 1: AWS ロールを作成する。
- *codecatalyst-artifact-bucket* で作成した Amazon S3 の名前を入力しますステップ 2: Amazon S3 バケットを作成する。

このファイルのプロパティの詳細については、「」を参照してくださいビルドおよびテストアクションの YAML 定義。

5. (オプション) 検証 を選択して、コミットする前に YAML コードが有効であることを確認します。
6. [Commit] (コミット) を選択します。
7. コミットワークフローダイアログボックスで、次のように入力します。

   a. ワークフローファイル名 の場合、デフォルトの のままにしますcodecatalyst-artifact-workflow。

   b. コミットメッセージ には、次のように入力します。

```
add initial workflow file
```

c. リポジトリ で、 を選択しますcodecatalyst-artifact-source-repository。

d. ブランチ名 で、メイン を選択します。

e. [Commit] (コミット) を選択します。

これでワークフローが作成されました。ワークフロー実行は、ワークフローの上部で定義されているトリガーが原因で自動的に開始されます。具体的には、`codecatalyst-artifact-workflow.yaml`ファイルをソースリポジトリにコミット (およびプッシュ) すると、トリガーによってワークフロー実行が開始されます。

進行中のワークフロー実行を表示するには

1. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
2. 先ほど作成したワークフローを選択します: `codecatalyst-artifact-workflow`。
3. 最初のビルドアクションの進行状況GenerateFilesを表示するには、 を選択します。
4. アップロードを選択して、2 番目のビルドアクションの進行状況を確認します。
5. アップロードアクションが終了したら、次の操作を行います。

   - ワークフローの実行が成功したら、次の手順に進みます。
   - ワークフローの実行に失敗した場合は、ログを選択して問題をトラブルシューティングします。

ステップ 5: 結果を確認する

ワークフローが実行されたら、Amazon S3 サービスに移動して*`codecatalyst-artifact-bucket`*バケットを確認します。これで、次のファイルとフォルダが含まれるはずです。

```
.
|– .aws/
|– .git/
|Goodbye.txt
|Hello.txt
|REAME.md
```

`Goodbye.txt` および `Hello.txt`ファイルはアーcodecatalystArtifactティファクトの一部であったため、アップロードされました。`.aws/`、`.git/`、および `README.md`ファイルは、ソースリポジトリにあったためアップロードされました。

クリーンアップ

これらのサービスに対して課金されない AWS ように、 CodeCatalyst および でクリーンアップします。

でクリーンアップするには CodeCatalyst

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. `codecatalyst-artifact-source-repository` ソースリポジトリを削除します。
3. `codecatalyst-artifact-workflow` ワークフローを削除します。

でクリーンアップするには AWS

1. Amazon S3 で次のようにクリーンアップします。
   a. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
   b. `codecatalyst-artifact-bucket` バケット内のファイルを削除します。
   c. `codecatalyst-artifact-bucket` バケットを削除します。
2. IAM で次のようにクリーンアップします。
   a. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
   b. `codecatalyst-s3-build-policy` を削除します。
   c. `codecatalyst-s3-build-role` を削除します。

ビルドおよびテストアクションの YAML 定義

以下は、ビルドアクションとテストアクションの YAML 定義です。YAML プロパティは非常に類似しているため、2 つのアクションには 1 つのリファレンスがあります。

このアクション定義は、より広範なワークフロー定義ファイル内のセクションとして存在します。ファイルの詳細については、「ワークフロー YAML 定義」を参照してください。

次のコードで YAML プロパティを選択すると、説明が表示されます。

> **Note**
>
> 後続の YAML プロパティのほとんどには、対応する UI 要素がビジュアルエディタにあります。UI 要素を検索するには、Ctrl+F を使用します。 要素は、関連付けられた YAML プロパティとともに一覧表示されます。

```
# The workflow definition starts here.
# See ######### for details.

Name: MyWorkflow
SchemaVersion: 1.0
Actions:

# The action definition starts here.
  action-name:
    Identifier: aws/build@v1 | aws/managed-test@v1
    DependsOn:
      - dependent-action-name-1
    Compute:
      Type: EC2 | Lambda
      Fleet: fleet-name
    Timeout: timeout-minutes
    Environment:
      Name: environment-name
      Connections:
        - Name: account-connection-name
          Role: iam-role-name
    Caching:
      FileCaching:
        key-name-1:
          Path: file1.txt
          RestoreKeys:
            - restore-key-1
    Inputs:
      Sources:
        - source-name-1
        - source-name-2
      Artifacts:
        - artifact-name
      Variables:
        - Name: variable-name-1
```

```
          Value: variable-value-1
        - Name: variable-name-2
          Value: variable-value-2
    Outputs:
      Artifacts:
        - Name: output-artifact-1
          Files:
            - build-output/artifact-1.jar
            - "build-output/build*"
        - Name: output-artifact-2
          Files:
            - build-output/artifact-2.1.jar
            - build-output/artifact-2.2.jar
      Variables:
        - variable-name-1
        - variable-name-2
      AutoDiscoverReports:
        Enabled: true | false
        ReportNamePrefix: AutoDiscovered
        IncludePaths:
          - "**/*"
        ExcludePaths:
          - node_modules/cdk/junit.xml
        SuccessCriteria:
          PassRate: percent
          LineCoverage: percent
          BranchCoverage: percent
          Vulnerabilities:
            Severity: CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL
            Number: whole-number
          StaticAnalysisBug:
            Severity: CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL
            Number: whole-number
          StaticAnalysisSecurity:
            Severity: CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL
            Number: whole-number
          StaticAnalysisQuality:
            Severity: CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL
            Number: whole-number
          StaticAnalysisFinding:
            Severity: CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL
            Number: whole-number
      Reports:
        report-name-1:
```

```
        Format: format
        IncludePaths:
          - "*.xml"
        ExcludePaths:
          - report2.xml
          - report3.xml
        SuccessCriteria:
          PassRate: percent
          LineCoverage: percent
          BranchCoverage: percent
          Vulnerabilities:
            Severity: CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL
            Number: whole-number
          StaticAnalysisBug:
              Severity: CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL
              Number: whole-number
          StaticAnalysisSecurity:
              Severity: CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL
              Number: whole-number
          StaticAnalysisQuality:
              Severity: CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL
              Number: whole-number
          StaticAnalysisFinding:
              Severity: CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL
              Number: whole-number
    Configuration:
      Container:
        Registry: registry
        Image: image
      Steps:
        - Run: "step 1"
        - Run: "step 2"
      Packages:
        NpmConfiguration:
          PackageRegistries:
            - PackagesRepository: package-repository
              Scopes:
                - "@scope"
```

action-name

(必須)

アクションの名前を指定します。すべてのアクション名は、ワークフロー内で一意である必要があります。アクション名は、英数字 (a～z、A～Z、0～9)、ハイフン (-)、アンダースコア (_) に制限されています。スペースは使用できません。引用符を使用してアクション名で特殊文字やスペースを有効にすることはできません。

対応する UI: 設定タブ/アクション名

Identifier

(*######* /Identifier)

アクションを識別します。バージョンを変更しない限り、このプロパティを変更しないでください。詳細については、「アクションのメジャー、マイナー、またはパッチバージョンの指定」を参照してください。

ビルドアクションaws/build@v1に を使用します。

テストアクションaws/managed-test@v1に を使用します。

対応する UI: ワークフロー図/アクション名/*aws/build@v1|aws/managed-test@v1* ラベル

DependsOn

(*######* /DependsOn)

(オプション)

このアクションを実行するために正常に実行する必要があるアクション、アクショングループ、またはゲートを指定します。

「依存」機能の詳細については、「」を参照してください他のアクションに依存するようにアクションを設定する。

対応する UI: の入力タブ/依存 - オプション

Compute

(*######* /Compute)

(オプション)

ワークフローアクションを実行するために使用されるコンピューティングエンジン。コンピューティングはワークフローレベルまたはアクションレベルで指定できますが、両方を指定することはできま

せん。ワークフローレベルで指定すると、コンピューティング設定はワークフローで定義されたすべてのアクションに適用されます。ワークフローレベルでは、同じインスタンスで複数のアクションを実行することもできます。詳細については、「アクション間でのコンピューティングの共有」を参照してください。

対応する UI: なし

Type

（ ######/Compute/タイプ ）

(Computeが含まれている場合は必須)

コンピューティングエンジンのタイプ。次のいずれかの値を使用できます。

- EC2 (ビジュアルエディタ) または EC2 (YAML エディタ)

  アクション実行中の柔軟性のために最適化されました。
- Lambda (ビジュアルエディタ) または Lambda (YAML エディタ)

  アクションの起動速度を最適化しました。

コンピューティングタイプの詳細については、「コンピューティングタイプ」を参照してください。

対応する UI: 設定タブ/コンピューティングタイプ

Fleet

（ ###### /Compute/Fleet ）

(オプション)

ワークフローまたはワークフローアクションを実行するマシンまたはフリートを指定します。オンデマンドフリートでは、アクションが開始されると、ワークフローは必要なリソースをプロビジョニングし、アクションが終了するとマシンは破棄されます。オンデマンドフリートの例: `Linux.x86-64.Large`、`Linux.x86-64.XLarge`。オンデマンドフリートの詳細については、「」を参照してくださいオンデマンドフリートのプロパティ。

プロビジョニングされたフリートでは、ワークフローアクションを実行するように一連の専用マシンを設定します。これらのマシンはアイドル状態のままで、すぐにアクションを処理できます。プロビ

ジョニングされたフリートの詳細については、「」を参照してくださいプロビジョニングされたフリートのプロパティ。

Fleet を省略した場合、デフォルトは ですLinux.x86-64.Large。

対応する UI: 設定タブ/コンピューティングフリート

Timeout

(###### /Timeout)

(オプション)

CodeCatalyst アクションが終了するまでに実行できる時間を分単位で指定します (YAML エディタ)、または時間と分単位で指定します (ビジュアルエディタ)。最小値は 5 分で、最大値は「」で説明されていますワークフローのクォータ。デフォルトのタイムアウトは、最大タイムアウトと同じです。

対応する UI: 設定タブ/タイムアウト - オプション

Environment

(###### /Environment)

(オプション)

アクションで使用する CodeCatalyst 環境を指定します。

環境の詳細については、環境を使用して AWS アカウント および VPCsにデプロイする CodeCatalyst「」および「」を参照してください環境を作成する。

対応する UI: 設定タブ/環境

Name

(###### /Environment/Name)

(オプション)

アクションに関連付ける既存の環境の名前を指定します。

対応する UI: 設定タブ/環境名

Connections

(###### /Environment/Connections)

(オプション)

アクションに関連付けるアカウント接続を指定します。で最大 1 つのアカウント接続を指定できますEnvironment。

アカウント接続の詳細については、「」を参照してください接続された AWS リソースへのアクセスを許可する AWS アカウント。アカウント接続を環境に関連付ける方法については、「」を参照してください環境を作成する。

対応する UI: 設定タブ/

Name

(###### /Environment/Connections/Name)

(オプション)

アカウント接続の名前を指定します。

対応する UI: 設定タブ/

Role

(###### /Environment/Connections/Role)

(オプション)

このアクションが Amazon S3 や Amazon ECR などの AWS サービスにアクセスして操作するために使用する IAM ロールの名前を指定します。 Amazon S3 このロールがアカウント接続に追加されていることを確認します。アカウント接続に IAM ロールを追加するには、「」を参照してくださいアカウント接続への IAM ロールの追加。

> **Note**
>
> 十分なアクセス許可があれば、ここでCodeCatalystWorkflowDevelopmentRole-*spaceName*ロールの名前を指定できます。このロールの詳細については、「アカウントとスペース

のCodeCatalystWorkflowDevelopmentRole-*spaceName*ロールの作成」を参照してください。CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールには、セキュリティリスクをもたらす可能性のある非常に広範なアクセス許可があることを理解します。このロールは、セキュリティが懸念されないチュートリアルとシナリオでのみ使用することをお勧めします。

> ⚠ Warning
>
> アクセス許可は、ビルドアクションとテストアクションに必要なアクセス許可に制限します。より広範なアクセス許可を持つロールを使用すると、セキュリティ上のリスクが生じる可能性があります。

対応する UI: 設定タブ/

Caching

(###### /Caching)

(オプション)

ディスク上のファイルを保存し、後続のワークフロー実行でそのキャッシュから復元するキャッシュを指定できるセクション。

ファイルキャッシュの詳細については、「」を参照してくださいワークフロー実行間のファイルのキャッシュ。

対応する UI: 設定タブ/ファイルのキャッシュ - オプション

FileCaching

(###### /Caching/FileCaching)

(オプション)

一連のキャッシュの設定を指定するセクション。

対応する UI: 設定タブ/ファイルキャッシュ - オプション/キャッシュの追加

key-name-1

(###### key-/Caching/FileCaching/***name-1***)

(オプション)

プライマリキャッシュプロパティ名の名前を指定します。キャッシュプロパティ名は、ワークフロー内で一意である必要があります。各アクションには、 に最大 5 つのエントリを含めることができますFileCaching。

対応する UI: 設定タブ/ファイルキャッシュ - オプション/キャッシュ/キーの追加

Path

(###### key-/Caching/FileCaching/*name-1*/Path)

(オプション)

キャッシュに関連するパスを指定します。

対応する UI: 設定タブ/ファイルキャッシュ - オプション/キャッシュ/パスの追加

RestoreKeys

(###### key-/Caching/FileCaching/*name-1*/RestoreKeys)

(オプション)

プライマリキャッシュプロパティが見つからない場合にフォールバックとして使用する復元キーを指定します。復元キー名はワークフロー内で一意である必要があります。各キャッシュには、 に最大 5 つのエントリを含めることができますRestoreKeys。

対応する UI: 設定タブ/ファイルキャッシュ - オプション/キャッシュの追加/キーの復元 - オプション

Inputs

(###### /Inputs)

(オプション)

Inputs セクションでは、ワークフローの実行中にアクションに必要なデータを定義します。

> **Note**
>
> ビルドアクションまたはテストアクションごとに最大 4 つの入力 (1 つのソースと 3 つのアーティファクト) が許可されます。変数はこの合計にはカウントされません。

異なる入力 (ソースとアーティファクトなど) にあるファイルを参照する必要がある場合、ソース入力はプライマリ入力、アーティファクトはセカンダリ入力です。セカンダリ入力内のファイルへの参照は、プライマリから区別するために特別なプレフィックスが付けられます。詳細については、「例: 複数のアーティファクト内のファイルを参照する」を参照してください。

対応する UI: Inputs タブ

Sources

(###### /Inputs/Sources)

(オプション)

アクションに必要なソースリポジトリを表すラベルを指定します。現在、サポートされているラベルは のみです。これは`WorkflowSource`、ワークフロー定義ファイルが保存されているソースリポジトリを表します。

ソースを省略する場合は、 で少なくとも 1 つの入力アーティファクトを指定する必要がありますaction-name`/Inputs/Artifacts`。

sources の詳細については、「ワークフローをソースリポジトリに接続する」を参照してください。

対応する UI: なし

Artifacts - input

(###### /Inputs/Artifacts)

(オプション)

このアクションへの入力として提供する以前のアクションのアーティファクトを指定します。これらのアーティファクトは、以前のアクションで出力アーティファクトとして定義しておく必要があります。

入力アーティファクトを指定しない場合は、 で少なくとも 1 つのソースリポジトリを指定する必要がありますaction-name`/Inputs/Sources`。

アーティファクトの詳細については、「」を参照してくださいアーティファクトを使用したワークフロー内のアクション間のデータの共有。

> **Note**
>
> Artifacts - オプションのドロップダウンリストが使用できない場合 (ビジュアルエディタ)、または YAML (YAML エディタ) の検証時に でエラーが発生した場合は、アクションが 1 つの入力のみをサポートしているためである可能性があります。この場合、ソース入力を削除してみてください。

対応する UI: 入力タブ/アーティファクト - オプション

Variables - input

(###### /Inputs/Variables)

(オプション)

アクションで使用できるようにしたい入力変数を定義する名前と値のペアのシーケンスを指定します。変数名は、英数字 (a ~ z、A ~ Z、0 ~ 9)、ハイフン (-)、アンダースコア (_) に制限されています。スペースは使用できません。変数名で特殊文字やスペースを有効にするために引用符を使用することはできません。

例を含む変数の詳細については、「」を参照してくださいワークフローでの変数の設定と使用。

対応する UI: 入力タブ/変数 - オプション

Outputs

(###### /Outputs)

(オプション)

ワークフローの実行中に アクションによって出力されるデータを定義します。

対応する UI: 出力タブ

Artifacts - output

(###### /Outputs/Artifacts)

(オプション)

アクションによって生成されたアーティファクトの名前を指定します。アーティファクト名はワークフロー内で一意である必要があり、英数字 (a～z、A～Z、0～9) とアンダースコア (_) に制限されます。スペース、ハイフン (-)、その他の特殊文字は使用できません。引用符を使用して、出力アーティファクト名でスペース、ハイフン、その他の特殊文字を有効にすることはできません。

アーティファクトの詳細については、「」を参照してくださいアーティファクトを使用したワークフロー内のアクション間のデータの共有。

対応する UI: 出力タブ/アーティファクト

Name

(###### /Outputs/Artifacts/Name)

(Artifacts - outputが含まれている場合は必須)

アクションによって生成されたアーティファクトの名前を指定します。アーティファクト名はワークフロー内で一意である必要があり、英数字 (a～z、A～Z、0～9) とアンダースコア (_) に制限されます。スペース、ハイフン (-)、その他の特殊文字は使用できません。引用符を使用して、出力アーティファクト名でスペース、ハイフン、その他の特殊文字を有効にすることはできません。

アーティファクトの詳細については、「」を参照してくださいアーティファクトを使用したワークフロー内のアクション間のデータの共有。

対応する UI: 出力タブ/アーティファクト/新しい出力/ビルドアーティファクト名

Files

(###### /Outputs/Artifacts/Files)

(Artifacts - outputが含まれている場合は必須)

アクションによって出力されるアーティファクトに CodeCatalyst を含むファイルを指定します。これらのファイルは、実行時にワークフローアクションによって生成され、ソースリポジトリでも使用できます。ファイルパスは、ソースリポジトリまたは前のアクションのアーティファクトに存在し、ソースリポジトリまたはアーティファクトルートを基準にしています。glob パターンを使用してパスを指定できます。例:

- ビルドまたはソースリポジトリの場所のルートにある 1 つのファイルを指定するには、`my-file.jar` を使用します。
- サブディレクトリ内の 1 つのファイルを指定するには、`directory/my-file.jar` または `directory/subdirectory/my-file.jar` を使用します。

- すべてのファイルを指定するには、"**/*" を使用します。glob パターン ** は、任意の数のサブディレクトリにマッチすることを示します。
- directory という名前のディレクトリ内のすべてのファイルとディレクトリを指定するには、"directory/**/*" を使用します。glob パターン ** は、任意の数のサブディレクトリにマッチすることを示します。
- directory という名前のディレクトリ内のすべてのファイルを指定するが、そのサブディレクトリを含めないようにするには、"directory/*" を使用します。

> **Note**
>
> ファイルパスに 1 つ以上のアスタリスク (*) またはその他の特殊文字が含まれている場合は、パスを二重引用符 () で囲みます""。特殊文字の詳細については、「」を参照してください構文のガイドラインと規則。

アーティファクトの詳細については、「」を参照してくださいアーティファクトを使用したワークフロー内のアクション間のデータの共有。

> **Note**
>
> ファイルパスにプレフィックスを追加して、検索するアーティファクトまたはソースを示す必要がある場合があります。詳細については、「ソースリポジトリ内のファイルを参照する」および「アーティファクト内のファイルを参照する」を参照してください。

対応する UI: タブ/アーティファクト/新しい出力/ビルドによって生成されたファイル

Variables - output

(###### /Outputs/Variables)

(オプション)

後続のアクションで使用できるように、アクションでエクスポートする変数を指定します。

例を含む変数の詳細については、「」を参照してくださいワークフローでの変数の設定と使用。

対応する UI: 出力タブ/変数/変数の追加

変数名 1

(###### *variable-*/Outputs/Variables/*name-1*)

(オプション)

アクションでエクスポートする変数の名前を指定します。この変数は、同じアクションの `Inputs`または `Steps`セクションで既に定義されている必要があります。

例を含む変数の詳細については、「」を参照してください[ワークフローでの変数の設定と使用]。

対応する UI: Outputs tab/Variables/Add variable/Name

AutoDiscoverReports

(###### /Outputs/AutoDiscoverReports)

(オプション)

自動検出機能の設定を定義します。

自動検出を有効にすると、 は、アクションに渡`Inputs`されたすべてのファイルと、アクション自体によって生成されたすべてのファイル CodeCatalyst を検索し、テスト、コードカバレッジ、ソフトウェアコンポジション分析 (SCA) レポートを探します。見つかったレポートごとに、 はレポートに変換 CodeCatalyst します CodeCatalyst 。CodeCatalyst レポートは、 CodeCatalyst サービスに完全に統合され、 CodeCatalyst コンソールから表示および操作できるレポートです。

> **Note**
>
> デフォルトでは、自動検出機能はすべてのファイルを検査します。IncludePaths または ExcludePathsプロパティを使用して、検査するファイルを制限できます。

対応する UI: タブ/レポート/自動検出レポートを出力する

Enabled

(###### /Outputs/AutoDiscoverReports/Enabled)

(オプション)

自動検出機能を有効または無効にします。

有効な値は `true` または `false` です。

`Enabled` を省略した場合、デフォルトは ですtrue。

対応する UI: タブ/レポート/自動検出レポートを出力する

ReportNamePrefix

(*######* /Outputs/AutoDiscoverReports/ReportNamePrefix)

(AutoDiscoverReportsが含まれ、有効になっている場合は必須)

関連するレポートに名前を付けるために、見つかったすべての CodeCatalyst レポートの CodeCatalyst 先頭に プレフィックスを指定します。例えば、プレフィックスとして を指定しAutoDiscovered CodeCatalyst、2 つのテストレポート `TestSuiteOne.xml`とを自動検出すると`TestSuiteTwo.xml`、関連する CodeCatalyst レポートの名前は `AutoDiscoveredTestSuiteOne`と になります`AutoDiscoveredTestSuiteTwo`。

対応する UI: 出力タブ/レポート/プレフィックス名

IncludePaths

(*######* /Outputs/AutoDiscoverReports/IncludePaths)

または

(*###### report-*/Outputs/Reports/*name-1*/IncludePaths)

(AutoDiscoverReportsが含まれ、有効になっている場合、または Reports が含まれている場合は必須)

raw レポートを検索するときに CodeCatalyst に含まれるファイルとファイルパスを指定します。例えば、 を指定すると`"/test/report/*"`、 は `/test/report/*` ディレクトリを検索するアクションで使用されるビルドイメージ全体 CodeCatalyst を検索します。そのディレクトリが見つか CodeCatalyst ると、 はそのディレクトリ内のレポートを検索します。

> **Note**
>
> ファイルパスに 1 つ以上のアスタリスク (*) またはその他の特殊文字が含まれている場合は、パスを二重引用符 () で囲みます`""`。特殊文字の詳細については、「」を参照してください構文のガイドラインと規則。

このプロパティを省略した場合、デフォルトは です。つまり"**/*"、検索にはすべてのパスのすべてのファイルが含まれます。

> **Note**
>
> 手動で設定するレポートの場合、 は単一のファイルに一致する glob パターンIncludePathsである必要があります。

対応する UI:

- 出力タブ/レポート/自動検出レポート/包含/除外パス/包含パス
- 出力タブ/レポート/レポートを手動で設定/*report-name-1*/Include/exclude パス/Include パス

ExcludePaths

(###### /Outputs/AutoDiscoverReports/ExcludePaths)

または

(###### report-/Outputs/Reports/name-1/ExcludePaths)

(オプション)

raw レポートを検索するときに が CodeCatalyst 除外するファイルとファイルパスを指定します。例えば、 を指定した場合"/test/my-reports/**/*"、 CodeCatalyst は /test/my-reports/ ディレクトリ内のファイルを検索しません。ディレクトリ内のすべてのファイルを無視するには、glob **/* パターンを使用します。

> **Note**
>
> ファイルパスに 1 つ以上のアスタリスク (*) またはその他の特殊文字が含まれている場合は、パスを二重引用符 () で囲みます""。特殊文字の詳細については、「」を参照してください構文のガイドラインと規則。

対応する UI:

- 出力タブ/レポート/自動検出レポート/包含/除外パス/除外パス

- 出力タブ/レポート/レポートを手動で設定/*report-name-1*/Include/exclude パス/Exclude パス

SuccessCriteria

(###### /Outputs/AutoDiscoverReports/SuccessCriteria)

または

(###### *report-*/Outputs/Reports/*name-1*/SuccessCriteria)

(オプション)

テスト、コードカバレッジ、ソフトウェア構成分析 (SCA)、静的分析 (SA) レポートの成功基準を指定します。

詳細については、「レポートの成功基準の設定」を参照してください。

対応する UI: 出力タブ/レポート/成功基準

PassRate

(###### /Outputs/AutoDiscoverReports/SuccessCriteria/PassRate)

または

(###### *report-*/Outputs/Reports/*name-1*/SuccessCriteria/PassRate)

(オプション)

関連する CodeCatalystレポートが合格としてマークされるようにするために合格する必要があるテストレポート内のテストの割合を指定します。有効な値には 10 進数が含まれます。例: 50、60.5。合格率基準はテストレポートにのみ適用されます。テストレポートの詳細については、「」を参照してくださいテストレポート。

対応する UI: 出力タブ/レポート/成功基準/合格率

LineCoverage

(###### /Outputs/AutoDiscoverReports/SuccessCriteria/LineCoverage)

または

(###### *report-*/Outputs/Reports/*name-1*/SuccessCriteria/LineCoverage)

(オプション)

コードカバレッジレポートで、関連するレポートが合格としてマークされる CodeCatalyst ためにカバーする必要がある行の割合を指定します。有効な値には 10 進数が含まれます。例: `50`、`60.5`。行カバレッジ基準は、コードカバレッジレポートにのみ適用されます。コードカバレッジレポートの詳細については、「」を参照してくださいコードカバレッジレポート。

対応する UI: 出力タブ/レポート/成功基準/ラインカバレッジ

BranchCoverage

(###### /Outputs/AutoDiscoverReports/SuccessCriteria/BranchCoverage)

または

(###### *report-*/Outputs/Reports/*name-1*/SuccessCriteria/BranchCoverage)

(オプション)

コードカバレッジレポートで、関連するレポートを合格としてマーク CodeCatalyst するためにカバーする必要があるブランチの割合を指定します。有効な値には 10 進数が含まれます。例: `50`、`60.5`。ブランチカバレッジ基準は、コードカバレッジレポートにのみ適用されます。コードカバレッジレポートの詳細については、「」を参照してくださいコードカバレッジレポート。

対応する UI: 出力タブ/レポート/成功基準/ブランチカバレッジ

Vulnerabilities

(###### /Outputs/AutoDiscoverReports/SuccessCriteria/Vulnerabilities)

または

(###### *report-*/Outputs/Reports/*name-1*/SuccessCriteria/Vulnerabilities)

(オプション)

SCA レポートで許可される脆弱性の最大数と重要度を指定して、関連する CodeCatalyst レポートが合格としてマークされるようにします。脆弱性を指定するには、以下を指定する必要があります。

- カウントに含める脆弱性の最小重要度。最も重要度の高い値から低い値までの有効な値は、`CRITICAL`、`HIGH`、、`MEDIUMLOW`、 です`INFORMATIONAL`。

例えば、 を選択するとHIGH、 HIGHと のCRITICAL脆弱性が集計されます。

- 許可する指定された重要度の脆弱性の最大数。この数を超えると、 CodeCatalyst レポートは失敗としてマークされます。有効な値は整数です。

脆弱性基準は SCA レポートにのみ適用されます。SCA レポートの詳細については、「」を参照してくださいソフトウェア構成分析レポート。

最小重要度を指定するには、 Severityプロパティを使用します。脆弱性の最大数を指定するには、 Numberプロパティを使用します。

対応する UI: 出力タブ/レポート/成功基準/脆弱性

StaticAnalysisBug

(###### /Outputs/AutoDiscoverReports/SuccessCriteria/StaticAnalysisBug)

または

(###### report-/Outputs/Reports/name-1/SuccessCriteria/StaticAnalysisBug)

(オプション)

SA レポートで許可されるバグの最大数と重要度を指定して、関連する CodeCatalyst レポートが合格としてマークされるようにします。バグを指定するには、以下を指定する必要があります。

- カウントに含めるバグの最小重要度。最も重要度の高い値から低い値までの有効な値は、CRITICAL、HIGH、、MEDIUMLOW、 ですINFORMATIONAL。

  例えば、 を選択するとHIGH、 HIGHと CRITICAL のバグが集計されます。
- 許可する指定された重要度のバグの最大数。この数を超えると、 CodeCatalyst レポートは失敗としてマークされます。有効な値は整数です。

バグ基準は、 PyLint および ESLint SA レポートにのみ適用されます。SA レポートの詳細については、「」を参照してください静的分析レポート。

最小重要度を指定するには、 Severityプロパティを使用します。脆弱性の最大数を指定するには、 Numberプロパティを使用します。

対応する UI: 出力タブ/レポート/成功基準/バグ

StaticAnalysisSecurity

(###### /Outputs/AutoDiscoverReports/SuccessCriteria/StaticAnalysisSecurity)

または

(###### report-/Outputs/Reports/name-1/SuccessCriteria/StaticAnalysisSecurity)

(オプション)

SA レポートで許可されるセキュリティ脆弱性の最大数と重要度を指定して、関連する CodeCatalyst レポートが合格としてマークされるようにします。セキュリティの脆弱性を指定するには、以下を指定する必要があります。

- カウントに含めるセキュリティ脆弱性の最小重要度。最も重要度の高い値から低い値までの有効な値は、CRITICAL、HIGH、、MEDIUMLOW、 ですINFORMATIONAL。

  例えば、 を選択するとHIGH、 HIGHと CRITICAL セキュリティの脆弱性が集計されます。
- 許可する指定された重要度のセキュリティ脆弱性の最大数。この数を超えると、 CodeCatalyst レポートは失敗としてマークされます。有効な値は整数です。

セキュリティの脆弱性基準は、 PyLint および ESLint SA レポートにのみ適用されます。SA レポートの詳細については、「」を参照してください静的分析レポート。

最小重要度を指定するには、 Severityプロパティを使用します。脆弱性の最大数を指定するには、 Numberプロパティを使用します。

対応する UI: 出力タブ/レポート/成功基準/セキュリティの脆弱性

StaticAnalysisQuality

(###### /Outputs/AutoDiscoverReports/SuccessCriteria/StaticAnalysisQuality)

または

(###### report-/Outputs/Reports/name-1/SuccessCriteria/StaticAnalysisQuality)

(オプション)

SA レポートで許可される品質問題の最大数と重要度を指定して、関連する CodeCatalyst レポートが合格としてマークされるようにします。品質問題を指定するには、以下を指定する必要があります。

- カウントに含める品質問題の最小重要度。最も重要度の高い値から低い値までの有効な値は、CRITICAL、HIGH、、MEDIUMLOW、 ですINFORMATIONAL。

  例えば、 を選択するとHIGH、 HIGHおよび CRITICAL品質の問題が集計されます。
- 許可する指定された重要度の品質問題の最大数。この数を超えると、 CodeCatalyst レポートは失敗としてマークされます。有効な値は整数です。

品質問題基準は、 PyLint および ESLint SA レポートにのみ適用されます。SA レポートの詳細については、「」を参照してください静的分析レポート。

最小重要度を指定するには、 Severityプロパティを使用します。脆弱性の最大数を指定するには、 Numberプロパティを使用します。

対応する UI: 出力タブ/レポート/成功基準/品質の問題

StaticAnalysisFinding

(###### /Outputs/AutoDiscoverReports/SuccessCriteria/StaticAnalysisFinding)

または

(###### *report-*/Outputs/Reports/*name-1*/SuccessCriteria/StaticAnalysisFinding)

(オプション)

SA レポートで許可される結果の最大数と重要度を指定して、関連する CodeCatalyst レポートが合格としてマークされるようにします。結果を指定するには、以下を指定する必要があります。

- カウントに含める結果の最小重要度。最も重要度の高い値から低い値までの有効な値は、CRITICAL、HIGH、、MEDIUMLOW、 ですINFORMATIONAL。

  例えば、 を選択するとHIGH、 HIGHと CRITICALの結果が集計されます。
- 許可する指定された重要度の検出結果の最大数。この数を超えると、 CodeCatalyst レポートは失敗としてマークされます。有効な値は整数です。

検出結果は SARIF SA レポートにのみ適用されます。SA レポートの詳細については、「」を参照してください静的分析レポート。

最小重要度を指定するには、 Severityプロパティを使用します。脆弱性の最大数を指定するには、 Numberプロパティを使用します。

対応する UI: 出力タブ/レポート/成功基準/検出結果

Reports

(######/Outputs/Reports)

(オプション)

テストレポートの設定を指定するセクション。

対応する UI: 出力タブ/レポート

report-name-1

(###### report-/Outputs/Reports/name-1)

(Reportsが含まれている場合は必須)

raw CodeCatalyst レポートから生成されるレポートに付ける名前。

対応する UI: 出力タブ/レポート/レポートを手動で設定/レポート名

Format

(###### report-/Outputs/Reports/name-1/Format)

(Reportsが含まれている場合は必須)

レポートに使用するファイル形式を指定します。指定できる値は以下のとおりです。

- テストレポートの場合：
  - Cucumber JSON には、Cucumber (ビジュアルエディタ) または CUCUMBERJSON (YAML エディタ) を指定します。
  - JUnit XML の場合は、JUnit (ビジュアルエディタ) または JUNITXML (YAML エディタ) を指定します。
  - NUnit XML の場合は、NUnit (ビジュアルエディタ) または NUNITXML (YAML エディタ) を指定します。
  - NUnit 3 XML の場合は、NUnit3 (ビジュアルエディタ) または NUNIT3XML (YAML エディタ) を指定します。

  - Visual Studio TRX には、Visual Studio TRX (ビジュアルエディタ) または VISUALSTUDIOTRX (YAML エディタ) を指定します。
  - TestNG XML には、TestNG (ビジュアルエディタ) または TESTNGXML (YAML エディタ) を指定します。
- コードカバレッジレポートの場合：
  - クローバー XML には、クローバー (ビジュアルエディタ) または CLOVERXML (YAML エディタ) を指定します。
  - コベルチュラ XML では、コベルチュラ (ビジュアルエディタ) または COBERTURAXML (YAML エディタ) を指定します。
  - JaCoCo XML の場合は、 JaCoCo (ビジュアルエディタ) または JACOCOXML (YAML エディタ) を指定します。
  - simplecov-json ではなく simplecov によって生成された SimpleCov JSON の場合は、Simplecov (ビジュアルエディタ) または SIMPLECOV (YAML エディタ) を指定します。 https://github.com/vicentllongo/simplecov-json
- ソフトウェアコンポジション分析 (SCA) レポートの場合：
  - SARIF には、SARIF (ビジュアルエディタ) または SARIFSCA (YAML エディタ) を指定します。

対応する UI: 出力タブ/レポート/レポートを手動で設定/レポートを追加/設定/*#####-1*/レポートタイプとレポート形式

Configuration

(*######* /Configuration)

(必須) アクションの設定プロパティを定義できるセクション。

対応する UI: 設定タブ

Container

(*######* /Configuration/Container)

(オプション)

アクションが処理を完了するために使用する Docker イメージまたはコンテナ を指定します。に付属するアクティブなイメージの 1 つを指定するか CodeCatalyst、独自のイメージを使用できます。独自のイメージを使用する場合は、Amazon ECR、Docker Hub、または別のレジストリに格納できま

す。Docker イメージを指定しない場合、アクションは処理にアクティブなイメージの 1 つを使用します。デフォルトで使用されているアクティブなイメージについては、「」を参照してくださいアクティブなイメージ。

独自の Docker イメージの指定の詳細については、「」を参照してくださいカスタムランタイム環境の Docker イメージを アクションに割り当てる。

対応する UI: ランタイム環境の Docker イメージ - オプション

Registry

(###### /Configuration/Container/Registry)

(Containerが含まれている場合は必須)

イメージが保存されているレジストリを指定します。有効な値を次に示します。

- CODECATALYST (YAML エディタ)

  イメージは CodeCatalyst レジストリに保存されます。
- Docker Hub (ビジュアルエディタ) または DockerHub (YAML エディタ)

  イメージは Docker Hub イメージレジストリに保存されます。
- その他のレジストリ (ビジュアルエディタ) または Other (YAML エディタ)

  イメージはカスタムイメージレジストリに保存されます。公開されているレジストリを使用することができます。
- Amazon Elastic Container Registry (ビジュアルエディタ) または ECR (YAML エディタ)

  イメージは Amazon Elastic Container Registry イメージリポジトリに保存されます。Amazon ECR リポジトリでイメージを使用するには、このアクションで Amazon ECR にアクセスする必要があります。このアクセスを有効にするには、次のアクセス許可とカスタム信頼ポリシーを含む IAM ロールを作成する必要があります。(既存のロールを変更して、必要に応じてアクセス許可とポリシーを含めることができます)。

  IAM ロールには、ロールポリシーに次のアクセス許可を含める必要があります。

  - ecr:BatchCheckLayerAvailability
  - ecr:BatchGetImage
  - ecr:GetAuthorizationToken

- ecr:GetDownloadUrlForLayer

IAM ロールには、次のカスタム信頼ポリシーを含める必要があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                 ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

IAM ロールの作成の詳細については、「IAM ユーザーガイド」の「カスタム信頼ポリシーを使用したロールの作成 (コンソール) 」を参照してください。

ロールを作成したら、 環境を介して アクションに割り当てる必要があります。詳細については、「環境、アカウント接続、IAM ロールをワークフローアクションに関連付ける」を参照してください。

対応する UI: Amazon Elastic Container Registry、Docker Hub、およびその他のレジストリオプション

Image

(###### /Configuration/Container/Image)

(Containerが含まれている場合は必須)

次のいずれかを指定します。

- CODECATALYST レジストリを使用している場合は、イメージ を次のアクティブなイメージ のいずれかに設定します。

- `CodeCatalystLinux_x86_64:2024_03`
- `CodeCatalystLinux_x86_64:2022_11`
- `CodeCatalystLinux_Arm64:2024_03`
- `CodeCatalystLinux_Arm64:2022_11`
- `CodeCatalystLinuxLambda_x86_64:2024_03`
- `CodeCatalystLinuxLambda_x86_64:2022_11`
- `CodeCatalystLinuxLambda_Arm64:2024_03`
- `CodeCatalystLinuxLambda_Arm64:2022_11`
- `CodeCatalystWindows_x86_64:2022_11`

- Docker Hub レジストリを使用している場合は、イメージを Docker Hub イメージ名とオプションのタグに設定します。

  例: `postgres:latest`
- Amazon ECR レジストリを使用している場合は、イメージを Amazon ECR レジストリ URI に設定します。

  例: `111122223333.dkr.ecr.us-west-2.amazonaws.com/codecatalyst-ecs-image-repo`
- カスタムレジストリを使用している場合は、イメージをカスタムレジストリで期待される値に設定します。

対応する UI: ランタイム環境の Docker イメージ (レジストリが の場合`CODECATALYST`)、Docker Hub イメージ (レジストリが Docker Hub の場合)、ECR イメージ URL (レジストリが Amazon Elastic Container Registry の場合)、イメージ URL (レジストリがその他のレジストリの場合)。

Steps

(`######` /Configuration/Steps)

(必須)

ビルドツールをインストール、設定、実行するアクション中に実行するシェルコマンドを指定します。

npm プロジェクトを構築する方法の例を次に示します。

```
Steps:
```

```
  - Run: npm install
  - Run: npm run build
```

ファイルパスを指定する方法の例を次に示します。

```
Steps:
  - Run: cd $ACTION_BUILD_SOURCE_PATH_WorkflowSource/app  && cat file2.txt
  - Run: cd $ACTION_BUILD_SOURCE_PATH_MyBuildArtifact/build-output/  && cat file.txt
```

ファイルパスの指定の詳細については、ソースリポジトリ内のファイルを参照する「」および「」を参照してくださいアーティファクト内のファイルを参照する。

対応する UI: 設定タブ/シェルコマンド

Packages

(###### /Configuration/Packages)

(オプション)

アクションが依存関係の解決に使用するパッケージリポジトリを指定できるセクション。パッケージを使用すると、アプリケーション開発に使用されるソフトウェアパッケージを安全に保存および共有できます。

パッケージの詳細については、「」を参照してくださいでソフトウェアパッケージを公開および共有する CodeCatalyst。

対応する UI: 設定タブ/パッケージ

NpmConfiguration

(###### /Configuration/Packages/NpmConfiguration)

(Packagesが含まれている場合は必須)

npm パッケージ形式の設定を定義するセクション。この設定は、ワークフロー実行中の アクションによって使用されます。

npm パッケージ設定の詳細については、「」を参照してくださいnpmを使う。

対応する UI: 設定タブ/パッケージ/設定の追加/npm

PackageRegistries

(###### /Configuration/Packages/NpmConfiguration/PackageRegistries)

(Packagesが含まれている場合は必須)

一連のパッケージリポジトリの設定プロパティを定義できるセクション。

対応する UI: 設定タブ/パッケージ/設定の追加/npm/パッケージリポジトリの追加

PackagesRepository

(###### /Configuration/Packages/NpmConfiguration/PackageRegistries/PackagesRepository)

(Packagesが含まれている場合は必須)

アクションで使用する CodeCatalyst パッケージリポジトリの名前を指定します。

複数のデフォルトリポジトリを指定すると、最後のリポジトリが優先されます。

パッケージリポジトリの詳細については、「」を参照してくださいパッケージリポジトリ。

対応する UI: 設定タブ/パッケージ/設定の追加/npm/パッケージリポジトリの追加/パッケージリポジトリ

Scopes

(###### /Configuration/Packages/NpmConfiguration/PackageRegistries/Scopes)

(オプション)

パッケージレジストリで定義するスコープのシーケンスを指定します。スコープを定義する場合、指定されたパッケージリポジトリは、リストされているすべてのスコープのレジストリとして設定されます。スコープを持つパッケージが npm クライアントを介してリクエストされた場合、デフォルトではなくそのリポジトリが使用されます。各スコープ名には「@」というプレフィックスを付ける必要があります。

スコープの上書きを含めると、最後のリポジトリが優先されます。

Scopes を省略すると、指定されたパッケージリポジトリが、 アクションで使用されるすべてのパッケージのデフォルトレジストリとして設定されます。

スコープの詳細については、パッケージ名前空間「」および「スコープ付きパッケージ」を参照してください。

対応する UI: 設定タブ/パッケージ/設定の追加/npm/パッケージリポジトリの追加/スコープ - オプション

ワークフローを使用したテスト

では CodeCatalyst、ビルドやテストなど、さまざまなワークフローアクションの一部としてテストを実行できます。これらのワークフローアクションはすべて品質レポートを生成できます。テストアクションは、テスト、コードカバレッジ、ソフトウェア構成分析、静的分析レポートを生成するワークフローアクションです。これらのレポートは CodeCatalyst コンソールに表示されます。

トピック

- 品質レポートタイプ
- テストアクションの追加
- テストアクションの結果の表示
- アクションで失敗したテストのスキップ
- テストアクション universal-test-runner への統合
- アクションでの品質レポートの設定
- レポートのテストケースの再試行
- でのテストのベストプラクティス CodeCatalyst
- ソフトウェア構成分析および静的分析レポートでサポートされる SARIF プロパティ

品質レポートタイプ

Amazon CodeCatalyst テストアクションは、次のタイプの品質レポートをサポートします。YAML でこれらのレポートをフォーマットする方法の例については、「」を参照してください品質レポート YAML の例。

トピック

- テストレポート
- コードカバレッジレポート
- ソフトウェア構成分析レポート

- 静的分析レポート

テストレポート

では CodeCatalyst、ビルド中に実行されるユニットテスト、統合テスト、およびシステムテストを設定できます。その後 CodeCatalyst 、 はテストの結果を含むレポートを作成できます。

テストレポートを使用すると、テストに関する問題のトラブルシューティングに役立ちます。複数のビルドからのテストレポートが多数ある場合は、テストレポートを使用して障害率を表示し、ビルドの最適化に役立てることができます。

次のテストレポートファイル形式を使用できます。

- Cucumber JSON (.json)
- JUnit XML (.xml)
- NUnit XML (.xml)
- NUnit3 XML (.xml)
- TestNG XML (.xml)
- Visual Studio TRX (.trx、.xml)

コードカバレッジレポート

では CodeCatalyst、テストのコードカバレッジレポートを生成できます。 CodeCatalyst は、次のコードカバレッジメトリクスを提供します。

ラインカバレッジ

テストがカバーするステートメントの数を測定します。ステートメントは、コメントを含まない単一の命令です。

```
line coverage = (total lines covered)/(total number of lines)
```

ブランチカバレッジ

や ifcaseステートメントなどのコントロール構造のすべての可能なブランチからテストがカバーするブランチの数を測定します。

```
branch coverage = (total branches covered)/(total number of branches)
```

以下のコードカバレッジレポートファイル形式がサポートされています。

- JaCoCo XML (.xml)
- SimpleCov JSON (Simplecov-json ではなく simplecov によって生成され、.json)
- クローバー XML (バージョン 3、.xml)
- コベルチュラ XML (.xml)
- LCOV (.info)

ソフトウェア構成分析レポート

では CodeCatalyst、ソフトウェアコンポジション分析 (SCA) ツールを使用してアプリケーションのコンポーネントを分析し、既知のセキュリティの脆弱性をチェックできます。重要度や修正方法が異なる脆弱性を詳細に説明する SARIF レポートを検出して解析できます。最も重要度の高いものから低いものの有効な重要度値は、CRITICAL、、HIGH、MEDIUMLOW、 ですINFORMATIONAL。

次の SCA レポートファイル形式がサポートされています。

- SARIF (.sarif、.json)

静的分析レポート

静的分析 (SA) レポートを使用して、ソースレベルのコードの欠陥を特定できます。では CodeCatalyst、SA レポートを生成して、デプロイ前にコードの問題を解決できます。これらの問題には、バグ、セキュリティの脆弱性、品質の問題、その他の脆弱性が含まれます。最も重要度の高いものから低いものの有効な重要度値は、CRITICAL、HIGH、、MEDIUMLOW、および ですINFORMATIONAL。

CodeCatalyst は、次の SA メトリクスを提供します。

バグ

ソースコードで見つかった可能性のあるバグの数を識別します。これらのバグには、メモリの安全性に関する問題が含まれる場合があります。バグの例を次に示します。

```
// The while loop will inadvertently index into array x out-of-bounds
int x[64];
while (int n = 0; n <= 64; n++) {
```

```
  x[n] = 0;
}
```

セキュリティの脆弱性

ソースコードで見つかった潜在的なセキュリティの脆弱性をいくつか特定します。これらのセキュリティの脆弱性には、シークレットトークンをプレーンテキストで保存するなどの問題が含まれる場合があります。

品質の問題

ソースコードで発生する可能性のある品質問題をいくつか特定します。これらの品質問題には、スタイル規則に関する問題が含まれる場合があります。品質問題の例を次に示します。

```
// The function name doesn't adhere to the style convention of camelCase
int SUBTRACT(int x, int y) {
  return x-y
}
```

その他の脆弱性

ソースコードで見つかった他の潜在的な脆弱性をいくつか特定します。

CodeCatalyst では、次の SA レポートファイル形式がサポートされています。

- PyLint (.py)
- ESLint (.js、.jsx、.ts、.tsx)
- SARIF (.sarif、.json)

テストアクションの追加

次の手順を使用して、 CodeCatalyst ワークフローにテストアクションを追加します。

Visual

ビジュアルエディタを使用してテストアクションを追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。

3. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
4. [編集] を選択します。
5. ビジュアル を選択します。
6. [アクション] を選択します。
7. アクション で、テスト を選択します。
8. 入力タブと設定タブで、必要に応じてフィールドに入力します。各フィールドの説明については、「」を参照してくださいビルドおよびテストアクションの YAML 定義。このリファレンスでは、YAML エディタとビジュアルエディタの両方に表示される各フィールド (および対応する YAML プロパティ値) に関する詳細情報を提供します。
9. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
10. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

YAML

YAML エディタを使用してビルドアクションを追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
3. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
4. [編集] を選択します。
5. YAML を選択します。
6. [アクション] を選択します。
7. アクション で、テスト を選択します。
8. 必要に応じて YAML コードのプロパティを変更します。使用可能な各プロパティの説明は、「」に記載されていますビルドおよびテストアクションの YAML 定義。
9. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
10. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

テストアクション定義

テストアクションは、ワークフロー定義ファイル内の一連の YAML プロパティとして定義されます。これらのプロパティの詳細については、ビルドおよびテストアクションの YAML 定義「」の「」を参照してくださいワークフロー YAML 定義。

テストアクションの結果の表示

生成されたログ、レポート、変数など、テストアクションの結果を表示するには、以下の手順に従います。

テストアクションの結果を表示するには

1. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
2. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
3. ワークフロー図で、テストアクションの名前を選択します。例えば、テスト です。
4. アクションによって生成されたログを表示するには、ログ を選択します。さまざまなアクションフェーズのログが表示されます。必要に応じて、ログを展開または折りたたむことができます。
5. テストアクションによって生成されたテストレポートを表示するには、レポート を選択するか、ナビゲーションペインでレポート を選択します。詳細については、「品質レポートタイプ」を参照してください。
6. テストアクションに使用される設定を表示するには、設定 を選択します。詳細については、「テストアクションの追加」を参照してください。
7. テストアクションで使用される変数を表示するには、変数 を選択します。詳細については、「ワークフローでの変数の設定と使用」を参照してください。

アクションで失敗したテストのスキップ

アクションに複数のテストコマンドがある場合は、前のコマンドが失敗した場合でも、アクション内の後続のテストコマンドの実行を許可できます。例えば、次のコマンドでは、 `test2`が`test1`失敗した場合でも、常に を実行したい場合があります。

```
Steps:
- Run: npm install
```

```
- Run: npm run test1
- Run: npm run test2
```

通常、ステップがエラーを返すと、Amazon はワークフローアクション CodeCatalyst を停止し、失敗としてマークします。エラー出力を にリダイレクトすることで、アクションステップの実行を継続できますnull。これを行うには、 コマンド2>/dev/nullに を追加します。この変更では、前の例は次のようになります。

```
Steps:
- Run: npm install
- Run: npm run test1 2>/dev/null
- Run: npm run test2
```

2 番目のコードスニペットでは、npm installコマンドのステータスは優先されますが、npm run test1コマンドによって返されるエラーは無視されます。その結果、 npm run test2 コマンドが実行されます。これにより、エラーが発生したかどうかにかかわらず、両方のレポートを一度に表示できます。

テストアクション universal-test-runner への統合

テストアクションは、オープンソースのコマンドラインツール と統合されますuniversal-test-runner。このツールは、テストレポートから 1 つ以上のテストケースを再試行するなどの高度なテスト機能を提供します。 universal-test-runnerは、テスト実行プロトコルを使用して、特定のフレームワーク内の任意の言語のテストを実行します。 は、次のフレームワークuniversal-test-runnerをサポートしています。

- Gradle
- ジェスト
- Maven
- pytest
- .NET

universal-test-runner は、テストアクション用にキュレーションされたイメージにのみインストールされます。カスタム Docker Hub または Amazon ECR を使用するようにテストアクションを設定する場合は、 を手動でインストールuniversal-test-runnerして高度なテスト機能を有効にする必要があります。そのためには、イメージに Node.js (14 以上) をインストールし、シェルコマンド npmを使用して universal-test-runnerから をインストールします- Run: npm

`install -g @aws/universal-test-runner`。シェルコマンドによるコンテナへの Node.js のインストールの詳細については、「Node Version Manager のインストールと更新」を参照してください。

の詳細についてはuniversal-test-runner、「 とは」を参照してください universal-test-runner。

Visual

ビジュアルエディタ universal-test-runner で を使用するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
3. ワークフローの名前を選択します。
4. [編集] を選択します。
5. ビジュアル を選択します。
6. [アクション] を選択します。
7. アクション で、テスト を選択します。
8. 設定タブで、サポートされているフレームワークを選択してサンプルコードを更新して、シェルコマンドフィールドに入力します。例えば、サポートされているフレームワークを使用するには、次のようなRunコマンドを使用します。

   ```
   - Run: run-tests <framework>
   ```

   必要なフレームワークがサポートされていない場合は、カスタムアダプターまたはランナーを提供することを検討してください。シェルコマンドフィールドの説明については、「」を参照してくださいSteps。
9. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
10. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

YAML

YAML エディタ universal-test-runner で を使用するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。

2. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
3. ワークフローの名前を選択します。
4. [編集] を選択します。
5. YAML を選択します。
6. [アクション] を選択します。
7. アクション で、テスト を選択します。
8. 必要に応じて YAML コードを変更します。例えば、サポートされているフレームワークを使用するには、次のようなRunコマンドを使用します。

```
Configuration:
  Steps:
    - Run: run-tests <framework>
```

必要なフレームワークがサポートされていない場合は、カスタムアダプターまたはランナーを提供することを検討してください。Steps プロパティの説明については、「」を参照してくださいSteps。

9. (オプション) Validate を選択して、コミットする前にワークフローの YAML コードを検証します。
10. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

アクションでの品質レポートの設定

このセクションでは、 アクションで品質レポートを設定する方法について説明します。

トピック

- 自動検出と手動レポート
- レポートの成功基準の設定
- 品質レポート YAML の例

自動検出と手動レポート

自動検出が有効になっている場合、 はアクションに渡されたすべての入力と、アクション自体によって生成されたすべてのファイル CodeCatalyst を検索し、テスト、コードカバレッジ、ソフトウェアコンポジション分析 (SCA)、静的分析 (SA) レポートを探します。これらの各レポートは、 で表示および操作できます CodeCatalyst。

生成されるレポートを手動で設定することもできます。生成するレポートのタイプとファイル形式を指定できます。詳細については、「品質レポートタイプ」を参照してください。

レポートの成功基準の設定

テスト、コードカバレッジ、ソフトウェアコンポジション分析 (SCA)、または静的分析 (SA) レポートの成功基準を決定する値を設定できます。

成功基準は、レポートが成功するか失敗するかを決定するしきい値です。 CodeCatalyst は、最初にテスト、コードカバレッジ、SCA、SA レポートなどのレポートを生成し、生成されたレポートに成功基準を適用します。次に、成功基準がどの程度満たされたかを示します。レポートが指定された成功基準を満たさない場合、成功基準を指定した CodeCatalyst アクションは失敗します。

例えば、SCA レポートの成功基準を設定する場合、最も重要度の高いものから最も重要度の低いものまでの有効な脆弱性値は`CRITICAL`、、`HIGH`、`MEDIUM`、`LOW`、 です`INFORMATIONAL`。HIGH 重要度が 1 つの脆弱性をスキャンするように基準を設定すると、`HIGH`重要度が 1 つ以上の脆弱性があるか、重要度が 1 つの脆弱性など、`HIGH`重要度がより高いレベルで少なくとも 1 つの脆弱性がある場合、レポートは失敗します`CRITICAL`。

成功基準を指定しない場合は、次のようになります。

- raw CodeCatalyst レポートに基づいて生成されたレポートには、成功基準は表示されません。
- 成功基準は、関連付けられたワークフローアクションが成功するか失敗するかを判断するために使用されません。

Visual

成功基準を設定するには

1. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
2. レポートを生成するアクションを含むワークフローを選択します。これは、成功基準を適用するレポートです。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
3. [編集] を選択します。
4. ビジュアル を選択します。
5. ワークフロー図で、 CodeCatalyst レポートを生成するように設定したアクションを選択します。

6. [出力] タブを選択します。
7. レポートの自動検出 または レポートを手動で設定 で、成功基準 を選択します。

   成功基準が表示されます。前の選択内容によっては、これらのオプションの一部またはすべてが表示される場合があります。

   パスレート

   関連する CodeCatalystレポートが合格としてマークされるようにするために合格する必要があるテストレポート内のテストの割合を指定します。有効な値には 10 進数が含まれます。例: 50、60.5。合格率基準はテストレポートにのみ適用されます。テストレポートの詳細については、「」を参照してくださいテストレポート。

   ラインカバレッジ

   コードカバレッジレポートで、関連するレポートが合格としてマークされる CodeCatalyst ためにカバーする必要がある行の割合を指定します。有効な値には 10 進数が含まれます。例: 50、60.5。行カバレッジ基準は、コードカバレッジレポートにのみ適用されます。コードカバレッジレポートの詳細については、「」を参照してくださいコードカバレッジレポート。

   ブランチカバレッジ

   コードカバレッジレポートで、関連するレポートが合格としてマークされる CodeCatalyst ためにカバーする必要があるブランチの割合を指定します。有効な値には 10 進数が含まれます。例: 50、60.5。ブランチカバレッジ基準は、コードカバレッジレポートにのみ適用されます。コードカバレッジレポートの詳細については、「」を参照してくださいコードカバレッジレポート。

   脆弱性 (SCA)

   SCA レポートで許可される脆弱性の最大数と重要度を指定して、関連する CodeCatalyst レポートが合格としてマークされるようにします。脆弱性を指定するには、以下を指定する必要があります。

   - カウントに含める脆弱性の最小重要度。最も重要度の高い値から低い値までの有効な値は、CRITICAL、HIGH、、MEDIUMLOW、 ですINFORMATIONAL。

     例えば、 を選択するとHIGH、 HIGHと のCRITICAL脆弱性が集計されます。

- 許可する指定された重要度の脆弱性の最大数。この数を超えると、 CodeCatalyst レポートは失敗としてマークされます。有効な値は整数です。

脆弱性基準は SCA レポートにのみ適用されます。SCA レポートの詳細については、「」を参照してくださいソフトウェア構成分析レポート。

バグ

SA レポートで許可されるバグの最大数と重要度を指定して、関連する CodeCatalyst レポートが合格としてマークされるようにします。バグを指定するには、以下を指定する必要があります。

- カウントに含めるバグの最小重要度。最も重要度の高い値から低い値までの有効な値は、CRITICAL、HIGH、、MEDIUMLOW、 ですINFORMATIONAL。

  例えば、 を選択するとHIGH、 HIGHと CRITICAL のバグが集計されます。
- 許可する指定された重要度のバグの最大数。この数を超えると、 CodeCatalyst レポートは失敗としてマークされます。有効な値は整数です。

バグ基準は、 PyLint および ESLint SA レポートにのみ適用されます。SA レポートの詳細については、「」を参照してください静的分析レポート。

セキュリティの脆弱性

SA レポートで許可されているセキュリティ脆弱性の最大数と重要度を指定し、関連する CodeCatalyst レポートが合格としてマークされるようにします。セキュリティの脆弱性を指定するには、以下を指定する必要があります。

- カウントに含めるセキュリティ脆弱性の最小重要度。最も重要度の高い値から低い値までの有効な値は、CRITICAL、HIGH、、MEDIUMLOW、 ですINFORMATIONAL。

  例えば、 を選択するとHIGH、 HIGHと CRITICAL セキュリティの脆弱性が集計されます。
- 許可する指定された重要度のセキュリティ脆弱性の最大数。この数を超えると、 CodeCatalyst レポートは失敗としてマークされます。有効な値は整数です。

セキュリティの脆弱性基準は、 PyLint および ESLint SA レポートにのみ適用されます。SA レポートの詳細については、「」を参照してください静的分析レポート。

品質の問題

SA レポートで許可される品質問題の最大数と重要度を指定して、関連する CodeCatalyst レポートが合格としてマークされるようにします。品質問題を指定するには、以下を指定する必要があります。

- カウントに含める品質問題の最小重要度。最も重要度の高い値から低い値までの有効な値は、CRITICAL、HIGH、、MEDIUMLOW、 ですINFORMATIONAL。

  例えば、 を選択するとHIGH、 HIGHと CRITICAL品質の問題が集計されます。
- 許可する指定された重要度の品質問題の最大数。この数を超えると、 CodeCatalyst レポートは失敗としてマークされます。有効な値は整数です。

品質問題基準は、 PyLint および ESLint SA レポートにのみ適用されます。SA レポートの詳細については、「」を参照してください静的分析レポート。

8. [Commit] (コミット) を選択します。
9. ワークフローを実行して、成功基準を raw レポート CodeCatalyst に適用し、成功基準情報を含む関連 CodeCatalyst レポートを再生成します。詳細については、「ワークフローの手動実行の開始」を参照してください。

YAML

成功基準を設定するには

1. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
2. レポートを生成するアクションを含むワークフローを選択します。これは、成功基準を適用するレポートです。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
3. [編集] を選択します。
4. YAML を選択します。
5. ワークフロー図で、 CodeCatalyst レポートを生成するように設定したアクションを選択します。
6. 詳細ペインで、出力タブを選択します。
7. アクション、 AutoDiscoverReportsセクション、または Reportsセクション で、SuccessCriteria、、、PassRate、LineCoverage、StaticAnalysisSecurity、

および プロパティとともに BranchCoverage Vulnerabilities StaticAnalysisBugStaticAnalysisQualityプロパティを追加します。

これらのプロパティの説明については、「」を参照してくださいビルドおよびテストアクションの YAML 定義。

8. [Commit] (コミット) を選択します。
9. ワークフローを実行して、未加工レポートに成功基準 CodeCatalyst を適用し、成功基準情報を含む関連する CodeCatalyst レポートを再生成します。ワークフローの開始の詳細については、「」を参照してくださいワークフローの手動実行の開始。

品質レポート YAML の例

次の例は、テストレポート、コードカバレッジレポート、ソフトウェア構成分析レポート、静的分析レポートの 4 つのレポートを手動で設定する方法を示しています。

```
Reports:
  MyTestReport:
    Format: JUNITXML
    IncludePaths:
      - "*.xml"
    ExcludePaths:
      - report1.xml
      SuccessCriteria:
        PassRate: 90
  MyCoverageReport:
    Format: CLOVERXML
    IncludePaths:
      - output/coverage/jest/clover.xml
      SuccessCriteria:
        LineCoverage: 75
        BranchCoverage: 75
  MySCAReport:
    Format: SARIFSCA
    IncludePaths:
      - output/sca/reports.xml
      SuccessCriteria:
        Vulnerabilities:
          Number: 5
          Severity: HIGH
  MySAReport:
    Format: ESLINTJSON
```

```
    IncludePaths:
      - output/static/eslint.xml
      SuccessCriteria:
        StaticAnalysisBug:
          Number: 10
          Severity: MEDIUM
        StaticAnalysisSecurity:
          Number: 5
          Severity: CRITICAL
        StaticAnalysisQuality:
          Number: 0
          Severity: INFORMATIONAL
```

レポートのテストケースの再試行

複数のテストケースが原因でレポートが失敗した場合、それらの個々のテストのみを再試行できます。これにより、テストケースの品質をすばやくチェックし、依存関係が壊れている、ワークフローの再実行を開始するなどの問題を解決するための次のステップを決定できます。テストアクションには、アクション全体ではなく、選択したテストケースのみを再試行`universal-test-runner`するが組み込まれています。一度に再試行できるテストケースは、アクションごとに 1 セットのみで、テストレポートごとに 5 回までです。詳細については、「テストアクション universal-test-runner への統合」を参照してください。

> **Note**
>
> レポートでテストケースを再試行しても、元のレポートを生成したワークフローのステータスには影響しません。

次の手順に従って、レポート内のテストケースを再試行します。

レポートのテストケースを再試行するには

1. ナビゲーションペインで [Reports] を選択します。
2. レポートの名前を選択します。レポートの名前、ステータス、リポジトリ、ブランチ、またはタイプでフィルタリングできます。
3. レポートの名前で、結果 を選択します。
4. 再試行するテストケースを選択し、「再実行」を選択し、「選択したテストケース」を選択します。

5. 再試行が完了したら、バナーで更新を選択し、更新された結果を表示します。

でのテストのベストプラクティス CodeCatalyst

が提供するテスト機能を使用する場合は CodeCatalyst、以下のベストプラクティスに従うことをお勧めします。

トピック

自動検出

でアクションを設定する場合 CodeCatalyst、自動検出を使用すると、JUnit テストレポートなどのさまざまなツールの出力を自動的に検出し、関連する CodeCatalystレポートを生成できます。自動検出は、検出された出力の名前やパスが変更されても、レポートが引き続き生成されるようにするのに役立ちます。新しいファイルが追加されると、 CodeCatalyst は自動的にそれらを検出し、関連するレポートを生成します。ただし、自動検出を使用する場合は、この機能の以下の側面の一部を考慮することが重要です。

- アクションで自動検出を有効にすると、同じタイプのすべての自動検出レポートが同じ成功基準を共有します。例えば、最小合格率などの共有基準は、自動検出されたすべてのテストレポートに適用されます。同じタイプのレポートに異なる基準が必要な場合は、これらの各レポートを明示的に設定する必要があります。
- 自動検出では、依存関係によって生成されたレポートも検索でき、成功基準が設定されている場合、これらのレポートに対する アクションが失敗する可能性があります。この問題は、除外パス設定を更新することで解決できます。
- 自動検出は実行時にアクションをスキャンするため、毎回同じレポートのリストを生成するとは限りません。特定のレポートを常に生成する場合は、レポートを明示的に設定する必要があります。例えば、ビルドの一部としてテストの実行を停止した場合、テストフレームワークは出力を生成せず、その結果、テストレポートが生成されず、アクションが成功する可能性があります。アクションの成功をその特定のテストに依存する場合は、そのレポートを明示的に設定する必要があります。

> **Tip**
>
> 新規または既存のプロジェクトを開始するときは、プロジェクトディレクトリ全体に対して自動検出を使用します (を含む**/*) 。これにより、サブディレクトリ内のファイルを含め、プロジェクト内のすべてのファイルでレポート生成が呼び出されます。

詳細については、「アクションでの品質レポートの設定」を参照してください。

成功基準

成功基準を設定することで、レポートに品質しきい値を適用できます。例えば、2 つのコードカバレッジレポートが自動検出された場合、1 つは 80% のラインカバレッジで、もう 1 つは 60% のラインカバレッジです。次のオプションがあります。

- ラインカバレッジの自動検出成功基準を 80% に設定します。これにより、最初のレポートが成功し、2 番目のレポートが失敗し、アクション全体が失敗します。ワークフローのブロックを解除するには、2 番目のレポートのラインカバレッジが 80% を超えるまで、プロジェクトに新しいテストを追加します。
- ラインカバレッジの自動検出成功基準を 60% に設定します。これにより、両方のレポートが成功し、アクションが成功します。その後、2 番目のレポートでコードカバレッジの拡大に取り組むことができます。ただし、この方法では、最初のレポートのカバレッジが 80% を下回っていないことを保証することはできません。
- ビジュアルエディタを使用するか、レポートごとに明示的な YAML セクションとパスを追加して、レポートの一方または両方を明示的に設定します。これにより、レポートごとに個別の成功基準とカスタム名を設定できます。ただし、この方法では、レポートパスが変更されるとアクションが失敗する可能性があります。

詳細については、「レポートの成功基準の設定」を参照してください。

パスの包含/除外

アクションの結果を確認するときは、 IncludePathsと を設定 CodeCatalyst することで、 によって生成されるレポートのリストを調整できますExcludePaths。

- を使用してIncludePaths、レポートを検索するときに含め CodeCatalyst るファイルとファイルパスを指定します。例えば、 を指定すると"/test/report/*"、 は /test/report/ ディレ

クトリを検索するアクションで使用されるビルドイメージ全体 CodeCatalyst を検索します。そのディレクトリが見つか CodeCatalyst ると、 はそのディレクトリ内のレポートを検索します。

> **Note**
>
> 手動で設定するレポートの場合、 は 1 つのファイルに一致する glob パターンIncludePathsである必要があります。

- を使用してExcludePaths、レポートの検索時に除外 CodeCatalyst するファイルとファイルパスを指定します。例えば、 を指定した場合"/test/reports/**/*"、 CodeCatalyst は /test/reports/ ディレクトリ内のファイルを検索しません。ディレクトリ内のすべてのファイルを無視するには、glob **/* パターンを使用します。

以下は、可能な glob パターンの例です。

パターン	説明
.	ドットを含む現在のディレクトリ内のすべてのオブジェクト名に一致します
*.xml	で終わる現在のディレクトリ内のすべてのオブジェクト名を一致 .xml
*.{xml,txt}	.xml または で終わる現在のディレクトリ内のすべてのオブジェクト名に一致します .txt
**/*.xml	で終わるすべてのディレクトリのオブジェクト名を一致 .xml
testFolder	というオブジェクトをファイルとして扱いtestFolder 、一致させます。
testFolder/*	testFolder などの からサブフォルダの 1 つのレベルのオブジェクトを照合します。testFolder/file.xml
testFolder/*/*	サブフォルダの 2 つのレベルのオブジェクトを から一致させますtestFolder 。

パターン	説明
	testFolder/reportsFolder/file.xml
testFolder/**	testFolder/file.xml や testFolder/otherFolder/file.xml などの testFolder 以下のファイルと同様に、サブフォルダ testFolder と一致する

CodeCatalyst は glob パターンを次のように解釈します。

- スラッシュ (/) 文字は、ファイルパス内のディレクトリを区切ります。
- アスタリスク (*) 記号は、フォルダの境界を超えない、0 文字以上の名前の要素と一致します。
- 二重アスタリスク (**) は、すべてのディレクトリをまたいで、0 文字以上の名前の要素と一致します。

> **Note**
>
> ExcludePaths は よりも優先されますIncludePaths。IncludePaths と の両方に同じフォルダExcludePathsが含まれている場合、そのフォルダはレポート用にスキャンされません。

ソフトウェア構成分析および静的分析レポートでサポートされる SARIF プロパティ

SARIF (統計分析結果交換形式) は、 のソフトウェアコンポジション分析および静的分析レポートで使用できる出力ファイル形式です CodeCatalyst。次の例は、静的分析レポートで SARIF を手動で設定する方法を示しています。

```
Reports:
MySAReport:
Format: SARIFSA
IncludePaths:
    - output/sa_report.json
SuccessCriteria:
```

```
    StaticAnalysisFinding:
    Number: 25
    Severity: HIGH
```

CodeCatalyst では、分析結果がレポートにどのように表示されるかを最適化するために使用できる次の SARIF プロパティがサポートされています。

トピック

- sarifLog オブジェクト
- run オブジェクト
- toolComponent オブジェクト
- reportingDescriptor オブジェクト
- result オブジェクト
- location オブジェクト
- physicalLocation オブジェクト
- logicalLocation オブジェクト
- fix オブジェクト

`sarifLog` オブジェクト

名前	必要	説明
`$schema`	はい	バージョン 2.1.0 の SARIF JSON スキーマの URI。
`version`	はい	CodeCatalyst は SARIF バージョン 2.1.0 のみをサポートします。
`runs[]`	はい	SARIF ファイルには 1 つ以上の実行の配列が含まれており、それぞれが分析ツールの 1 回の実行を表します。

run オブジェクト

名前	必要	説明
tool.driver	はい	分析ツールを記述するtoolComponent オブジェクト。
tool.name	いいえ	分析の実行に使用されるツールの名前を示すプロパティ。
results[]	はい	に表示される分析ツールの結果 CodeCatalyst。

toolComponent オブジェクト

名前	必要	説明
name	はい	分析ツールの名前。
properties.artifactScanned	いいえ	ツールによって分析されたアーティファクトの合計数。
rules[]	はい	ルールを表すreporting Descriptor オブジェクトの配列。これらのルールに基づいて、分析ツールは分析されるコードで問題を検出します。

reportingDescriptor オブジェクト

名前	必要	説明
id	はい	結果の参照に使用されるルールの一意の識別子。

名前	必要	説明
		最大長: 1,024 文字
name	いいえ	ルールの表示名。 最大長: 1,024 文字
shortDescription.text	いいえ	ルールの短縮された説明。 最大長: 3,000 文字
fullDescription.text	いいえ	ルールの完全な説明。 最大長: 3,000 文字
helpUri	いいえ	ルールのプライマリドキュメントの絶対 URI を含むようにローカライズできる文字列。 最大長: 3,000 文字
properties.unscore	いいえ	スキャン結果がスコアリングされたかどうかを示すフラグ。
properties.score.severity	いいえ	結果の重要度レベルを指定する固定文字列セット。 最大長: 1,024 文字
properties.cvssv3_baseSeverity	いいえ	Common Vulnerability Scoring System v3.1 の定性的重要度評価。
properties.cvssv3_baseScore	いいえ	CVSS v3 ベーススコアは 0.0 ~ 10.0 の範囲です。
properties.cvssv2_severity	いいえ	CVSS v3 値が使用できない場合、 は CVSS v2 値 CodeCatalyst を検索します。

名前	必要	説明
properties.cvssv2_score	いいえ	0.0 ~ 10.0 の範囲の CVSS v2 ベーススコア。
properties.severity	いいえ	結果の重要度レベルを指定する固定文字列セット。 最大長: 1,024 文字
defaultConfiguration.level	いいえ	ルールのデフォルトの重要度。

result オブジェクト

名前	必要	説明
ruleId	はい	結果の参照に使用されるルールの一意の識別子。 最大長: 1,024 文字
ruleIndex	はい	ツールコンポーネント 内の関連付けられたルールのインデックスrules[]。
message.text	はい	結果を記述し、各結果のメッセージを表示するメッセージ。 最大長: 3,000 文字
rank	いいえ	結果の優先度または重要度を表す 0.0 ~ 100.0 の値。このスケール値 0.0 が最も優先度が低く、100.0 が最も優先度が高いです。
level	いいえ	結果の重要度。

名前	必要	説明
		最大長: 1,024 文字
properties.unscore	いいえ	スキャン結果がスコアリングされたかどうかを示すフラグ。
properties.score.severity	いいえ	結果の重要度レベルを指定する固定文字列セット。 最大長: 1,024 文字
properties.cvssv3_baseSeverity	いいえ	Common Vulnerability Scoring System v3.1 の定性的重要度評価。
properties.cvssv3_baseScore	いいえ	0.0 ~ 10.0 の範囲の CVSS v3 ベーススコア。
properties.cvssv2_severity	いいえ	CVSS v3 値が使用できない場合、 は CVSS v2 値 CodeCatalyst を検索します。
properties.cvssv2_score	いいえ	0.0 ~ 10.0 の範囲の CVSS v2 ベーススコア。
properties.severity	いいえ	結果の重要度レベルを指定する固定文字列セット。 最大長: 1,024 文字

名前	必要	説明
locations[]	はい	結果が検出された場所のセット。指定されたすべての場所で変更を行うことによってのみ問題を修正できる場合を除き、1 つの場所のみを含める必要があります。 CodeCatalyst は、位置配列の最初の値を使用して結果に注釈を付けます。 location オブジェクトの最大数: 10
relatedLocations[]	いいえ	検出結果内の追加の口ケーションリファレンスのリスト。 location オブジェクトの最大数: 50
fixes[]	いいえ	スキャンツールによって提供されるレコメンデーションを表すfixオブジェクトの配列。 はfixes、配列内の最初のレコメンデーション CodeCatalyst を使用します。

location オブジェクト

名前	必要	説明
physicalLocation	はい	アーティファクトとリージョンを識別します。

名前	必要	説明
logicalLocations[]	いいえ	アーティファクトを参照せずに名前で記述される場所のセット。

physicalLocation オブジェクト

名前	必要	説明
artifactLocation.uri	はい	アーティファクトの場所を示す URI。通常はリポジトリ内のファイル、またはビルド中に生成されたファイルです。
fileLocation.uri	いいえ	ファイルの場所を示すフォールバック URI。これは、 が空をartifactLocation.uri 返す場合に使用されます。
region.startLine	はい	リージョンの最初の文字の行番号。
region.startColumn	はい	リージョンの最初の文字の列番号。
region.endLine	はい	リージョンの最後の文字の行番号。
region.endColumn	はい	リージョン内の最後の文字の列番号。

`logicalLocation` オブジェクト

名前	必要	説明
`fullyQualifiedName`	いいえ	結果の場所を説明する追加情報。 最大長: 1,024 文字

`fix` オブジェクト

名前	必要	説明
`description.text`	いいえ	各結果のレコメンデーションを表示するメッセージ。 最大長: 3,000 文字
`artifactChanges.[0].artifactLocation.uri`	いいえ	更新する必要があるアーティファクトの場所を示す URI。

ワークフローを使用したデプロイ

CodeCatalyst ワークフロー を使用すると、アプリケーションやその他のリソースを Amazon ECS などのさまざまなターゲットにデプロイできます AWS Lambda。

アプリケーションをデプロイする方法

を通じてアプリケーションまたはリソースをデプロイするには CodeCatalyst、まずワークフローを作成し、その中にデプロイアクションを指定します。デプロイアクションは、デプロイする対象、デプロイする場所、デプロイ方法 (ブルー/グリーンスキームの使用など) を定義するワークフロー構成要素です。 CodeCatalyst コンソールのビジュアルエディタ、または YAML エディタを使用して、ワークフローにデプロイアクションを追加します。

アプリケーションまたはリソースをデプロイするための大まかな手順は次のとおりです。

アプリケーションをデプロイするには (高レベルタスク)

1. CodeCatalyst プロジェクトで、デプロイするアプリケーションのソースコードを追加します。詳細については、「のプロジェクトのリポジトリにソースコードを保存する CodeCatalyst」を参照してください。
2. CodeCatalyst プロジェクトでは、ワークフロー を作成します。ワークフローでは、アプリケーションを構築、テスト、デプロイする方法を定義します。詳細については、「ワークフローの開始方法」を参照してください。
3. ワークフローでは、トリガー 、ビルドアクション 、およびオプションでテストアクション を追加します。 詳細については、トリガーを使用したワークフローの自動実行の開始、ビルドアクションの追加、およびテストアクションの追加を参照してください。
4. ワークフローで、デプロイアクション を追加します。Amazon ECS など、さまざまなターゲットへのアプリケーションへの CodeCatalystが提供するデプロイアクションから選択できます。(ビルドアクションまたは GitHub アクションを使用してアプリケーションをデプロイすることもできます。 ビルドアクションと GitHub アクションの詳細については、「」を参照してくださいアクションをデプロイする代替方法。)
5. ワークフローは、手動またはトリガーを使用して自動的に開始します。ワークフローは、ビルド、テスト、デプロイのアクションを順番に実行して、アプリケーションとリソースをターゲットにデプロイします。詳細については、「ワークフローの手動実行の開始」を参照してください。

デプロイアクションのリスト

次のデプロイアクションを使用できます。

- AWS CloudFormation スタックのデプロイ — このアクションは、AWS Serverless Application Model指定したAWS CloudFormation テンプレート AWS に基づいて に CloudFormation スタックを作成します。詳細については、「ワークフローを使用した AWS CloudFormation スタックのデプロイ」を参照してください。
- Amazon ECS にデプロイする – このアクションは、指定したタスク定義ファイルを登録します。詳細については、「ワークフローを使用した Amazon Elastic Container Service (ECS) へのアプリケーションのデプロイ」を参照してください。
- Kubernetes クラスターにデプロイ – このアクションは、アプリケーションを Amazon Elastic Kubernetes Service クラスターにデプロイします。詳細については、「ワークフローを使用した Amazon Elastic Kubernetes Service へのアプリケーションのデプロイ」を参照してください。

- AWS CDK deploy – このアクションは、 にAWS CDK アプリケーションをデプロイします AWS。詳細については、「ワークフローを使用した AWS Cloud Development Kit (AWS CDK) アプリケーションのデプロイ」を参照してください。

> **Note**
>
> リソースをデプロイできる CodeCatalyst アクションは他にもありますが、デプロイ情報は環境ページに表示されないため、デプロイアクションとは見なされません。環境ページとデプロイの表示の詳細については、環境を使用して AWS アカウント および VPCsにデプロイする CodeCatalyst 「」および「」を参照してくださいデプロイステータス、コミット、プルリクエストの表示。

デプロイアクションの利点

ワークフロー内でデプロイアクションを使用すると、次の利点があります。

- デプロイ履歴 – デプロイの履歴を表示して、デプロイされたソフトウェアの変更を管理および通信できるようにします。
- トレーサビリティ — CodeCatalyst コンソールを使用してデプロイのステータスを追跡し、各アプリケーションリビジョンがいつ、どこでデプロイされたかを確認します。
- ロールバック — エラーが発生した場合は、デプロイを自動的にロールバックします。デプロイのロールバックをアクティブ化するようにアラームを設定することもできます。
- モニタリング — ワークフローのさまざまな段階を進行するデプロイを監視します。
- 他の CodeCatalyst 機能との統合 – ソースコードを保存し、1 つのアプリケーションから構築、テスト、デプロイします。

アクションをデプロイする代替方法

デプロイアクションを使用する必要はありませんが、前のセクションで説明した利点が得られるため、デプロイアクションが推奨されます。代わりに、次のCodeCatalyst アクションを使用できます。

- ビルドアクション。

通常、対応するデプロイアクションが存在しないターゲットにデプロイする場合、またはデプロイ手順をより詳細に制御する場合は、ビルドアクションを使用します。ビルドアクションを使用してリソースをデプロイする方法の詳細については、「」を参照してくださいワークフローによる構築。

- GitHub アクション 。

  CodeCatalyst ワークフロー内でGitHub アクションを使用して、アプリケーションとリソースを (アクションの代わりに) CodeCatalystデプロイできます。 CodeCatalyst ワークフロー内で GitHub アクションを使用する方法については、「」を参照してください。 ワークフローへの GitHub アクションの統合

CodeCatalyst ワークフローを使用しない場合は、次の AWS サービスを使用してアプリケーションをデプロイすることもできます。

- AWS CodeDeploy – 「 とは」を参照してください CodeDeploy。
- AWS CodeBuild および AWS CodePipeline – 「 とは AWS CodeBuild」および「 とは」を参照してください AWS CodePipeline。
- AWS CloudFormation – 「 とは」を参照してください AWS CloudFormation。

複雑なエンタープライズデプロイには CodeDeploy CodeBuild CodePipeline、、、および CloudFormation サービスを使用します。

トピック

- ワークフローを使用した Amazon Elastic Container Service (ECS) へのアプリケーションのデプロイ
- ワークフローを使用した Amazon Elastic Kubernetes Service へのアプリケーションのデプロイ
- ワークフローを使用した AWS CloudFormation スタックのデプロイ
- ワークフローを使用した AWS Cloud Development Kit (AWS CDK) アプリケーションのデプロイ
- ワークフローを使用した AWS CDK アプリケーションのブートストラップ
- ワークフローを使用して Amazon S3 にファイルを発行する
- 環境を使用して AWS アカウント および VPCsにデプロイする CodeCatalyst
- ワークフロー図にデプロイされたアプリケーションの URL を表示する
- デプロイターゲットの削除

- コミットによるデプロイステータスの追跡
- デプロイログの表示
- デプロイステータス、コミット、プルリクエストの表示

ワークフローを使用した Amazon Elastic Container Service (ECS) へのアプリケーションのデプロイ

このセクションでは、 CodeCatalyst ワークフローを使用してコンテナ化されたアプリケーションを Amazon ECS クラスターにデプロイする方法について説明します。これを実現するには、Amazon ECS へのデプロイアクションをワークフローに追加する必要があります。このアクションは、指定したタスク定義ファイルを登録します。登録時に、タスク定義は Amazon ECS クラスター で実行されている Amazon ECS サービスによってインスタンス化されます。 https://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html#welcome-clusters「タスク定義のインスタンス化」は、アプリケーションを Amazon ECS にデプロイすることと同じです。

このアクションを使用するには、Amazon ECS クラスター、サービス、およびタスク定義ファイルを準備しておく必要があります。

Amazon ECS の詳細については、「Amazon Elastic Container Service デベロッパーガイド」を参照してください。

> **Tip**
>
> Amazon ECS へのデプロイアクションの使用方法を示すチュートリアルについては、「」を参照してくださいチュートリアル: Amazon ECS にアプリケーションをデプロイする。

> **Tip**
>
> Amazon ECS にデプロイアクションの実例については、 で Node.js API AWS Fargate を使用するか、設計図で Java API を使用して AWS Fargateプロジェクトを作成します。詳細については、「設計図を使用したプロジェクトの作成」を参照してください。

トピック

- チュートリアル: Amazon ECS にアプリケーションをデプロイする

- 「Amazon ECS へのデプロイ」アクションの追加
- 「Amazon ECS へのデプロイ」アクションによって生成される変数
- 「Amazon ECS にデプロイ」アクション YAML 定義

チュートリアル: Amazon ECS にアプリケーションをデプロイする

このチュートリアルでは、ワークフロー、Amazon ECS、および他のいくつかのサービスを使用して、サーバーレスアプリケーションを Amazon Elastic Container Service (Amazon ECS) にデプロイする方法について説明します AWS 。デプロイされたアプリケーションは、Apache ウェブサーバーの Docker イメージ上に構築されたシンプルな Hello World ウェブサイトです。このチュートリアルでは、クラスターの設定など、必要な準備作業を順を追って説明し、アプリケーションを構築およびデプロイするワークフローを作成する方法について説明します。

> **Tip**
>
> このチュートリアルを進める代わりに、完全な Amazon ECS セットアップを実行するブループリントを使用できます。Node.js API AWS Fargate を で、または Java API を AWS Fargateブループリントで使用する必要があります。詳細については、「設計図を使用したプロジェクトの作成」を参照してください。

トピック

- 前提条件
- ステップ 1: AWS ユーザーと を設定する AWS CloudShell
- ステップ 2: プレースホルダーアプリケーションを Amazon ECS にデプロイする
- ステップ 3: Amazon ECR イメージリポジトリを作成する
- ステップ 4: AWS ロールを作成する
- ステップ 5: に AWS ロールを追加する CodeCatalyst
- ステップ 6: ソースリポジトリを作成する
- ステップ 7: ソースファイルを追加する
- ステップ 8: ワークフローを作成して実行する
- ステップ 9: ソースファイルを変更する
- クリーンアップ

前提条件

開始する前に:

- 接続された AWS アカウントを持つ CodeCatalyst スペースが必要です。詳細については、「スペースの作成」を参照してください。
- スペースには、空の Start from scratch CodeCatalystプロジェクトが必要です。

  ```
  codecatalyst-ecs-project
  ```

  詳細については、「Amazon での空のプロジェクトの作成 CodeCatalyst」を参照してください。
- プロジェクトには、次の CodeCatalyst 環境が必要です。

  ```
  codecatalyst-ecs-environment
  ```

  この環境を次のように設定します。
  - 非本番環境の など、任意のタイプを選択します。
  - アカウントをその AWS アカウントに接続します。

  詳細については、「環境を使用して AWS アカウント および VPCsにデプロイする CodeCatalyst」を参照してください。

ステップ 1: AWS ユーザーと を設定する AWS CloudShell

このチュートリアルの最初のステップは、 でユーザーを作成し AWS IAM Identity Center、このユーザーとしてインスタンスを起動 AWS CloudShell することです。このチュートリアルの期間中、 CloudShell は開発用コンピュータであり、 は AWS リソースとサービスを設定する場所です。チュートリアルを完了したら、このユーザーを削除します。

> **Note**
>
> このチュートリアルでは、ルートユーザーを使用しないでください。別のユーザーを作成する必要があります。作成しないと、後で AWS Command Line Interface (CLI) でアクションを実行するときに問題が発生する可能性があります。

IAM Identity Center ユーザーおよび の詳細については CloudShell、「 AWS IAM Identity Center ユーザーガイド」および「 ユーザーガイドAWS CloudShell 」を参照してください。

IAM Identity Center ユーザーを作成するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/singlesignon/ で AWS IAM Identity Center コンソールを開きます。

   > **Note**
   >
   > スペース AWS アカウント に接続されている CodeCatalyst を使用してサインインしていることを確認してください。接続されているアカウントを確認するには、スペースに移動し、AWS アカウントタブを選択します。詳細については、「スペースの作成」を参照してください。

2. ナビゲーションペインで [Users] (ユーザー)、[Add user] (ユーザーの追加) の順に選択します。
3. ユーザー名 で、次のように入力します。

   ```
   CodeCatalystECSUser
   ```

4. パスワード で、このユーザー と共有できるワンタイムパスワードの生成 を選択します。
5. E メールアドレス と E メールアドレスの確認 で、IAM Identity Center にまだ存在しない E メールアドレスを入力します。
6. 名 と姓 で、次のように入力します。

   ```
   CodeCatalystECSUser
   ```

7. 表示名 で、自動的に生成された名前を保持します。

   ```
   CodeCatalystECSUser CodeCatalystECSUser
   ```

8. [次へ] をクリックします。
9. ユーザーをグループに追加ページで、次へ を選択します。
10. ユーザーの確認と追加ページで情報を確認し、ユーザーの追加 を選択します。

    ワンタイムパスワードダイアログボックスが表示されます。
11. アクセス AWS ポータル URL やワンタイムパスワードなどのサインイン情報をコピーして貼り付けます。
12. [閉じる] を選びます。

アクセス権限セットを作成するには

このアクセス許可セットは後で に割り当てCodeCatalystECSUserます。

1. ナビゲーションペインで [アクセス許可セット] を選択し、[アクセス許可セットの作成] を選択します。
2. 「事前定義されたアクセス許可セット」を選択し、「」を選択しますAdministratorAccess。このポリシーは、すべての に完全なアクセス許可を提供します AWS のサービス。
3. [次へ] をクリックします。
4. 「アクセス許可セット名」に、次のように入力します。

   ```
   CodeCatalystECSPermissionSet
   ```

5. [次へ] をクリックします。
6. [確認と作成] ページで情報を確認し、[グループの作成] を選択します。

アクセス許可セットを CodeCatalystECSUser に割り当てるには

1. ナビゲーションペインで を選択しAWS アカウント、現在サインイン AWS アカウント しているの横にあるチェックボックスをオンにします。
2. 「ユーザーまたはグループを割り当て」を選択します。
3. [ユーザー] タブを選択します。
4. [CodeCatalystECSUser] のチェックボックスをオンにします。
5. [次へ] をクリックします。
6. [CodeCatalystECSPermissionSet] のチェックボックスをオンにします。
7. [次へ] をクリックします。
8. 情報を確認し、[送信] を選択します。

   これで、 CodeCatalystECSUserと CodeCatalystECSPermissionSetを に割り当て AWS アカウント、それらをバインドしました。

サインアウトして CodeCatalystECSUser としてサインインし直すには

1. サインアウトする前に、 AWS アクセスポータル URL と、 のユーザー名とワンタイムパスワードがあることを確認してくださいCodeCatalystECSUser。以前にこの情報をテキストエディタにコピーしておく必要があります。

> **Note**
>
> この情報がない場合は、IAM Identity Center CodeCatalystECSUserの詳細ページに移動し、パスワードのリセット、ワンタイムパスワードの生成 [...]、パスワードのリセットをもう一度選択して、画面に情報を表示します。

2. からサインアウトします AWS。
3. AWS アクセスポータル URL をブラウザのアドレスバーに貼り付けます。
4. のユーザー名とワンタイムパスワードでサインインしますCodeCatalystECSUser。
5. 「新しいパスワード」でパスワードを入力し、「新しいパスワードを設定する」を選択します。

   画面に AWS アカウント ボックスが表示されます。
6. を選択しAWS アカウント、CodeCatalystECSUserユーザーとアクセス許可セット AWS アカウント を割り当てた の名前を選択します。
7. CodeCatalystECSPermissionSet の横にある [管理コンソール] を選択します。

   AWS Management Console が表示されます。これで、適切なアクセス許可CodeCatalystECSUserで としてサインインします。

AWS CloudShell インスタンスを起動するには

1. としてCodeCatalystECSUser、上部のナビゲーションバーで AWS アイコン () を選択します  。

   のメインページ AWS Management Console が表示されます。
2. 上部のナビゲーションバーで、 AWS CloudShell アイコン () を選択します  。

   CloudShell が開きます。 CloudShell 環境が作成されるまで待ちます。

> **Note**
>
> CloudShell アイコンが表示されない場合は、 でサポートされているリージョンCloudShellにいることを確認してください。このチュートリアルは、米国西部 (オレゴン) リージョンにいることを前提としています。

AWS CLI がインストールされていることを確認するには

1. CloudShell ターミナルで、次のように入力します。

```
aws --version
```

2. バージョンが表示されていることを確認します。

   AWS CLI は現在のユーザー 用にすでに設定されているためCodeCatalystECSUser、通常どおり、 AWS CLI キーと認証情報を設定する必要はありません。

ステップ 2: プレースホルダーアプリケーションを Amazon ECS にデプロイする

このセクションでは、プレースホルダーアプリケーションを Amazon ECS に手動でデプロイします。このプレースホルダーアプリケーションは、ワークフローによってデプロイされた Hello World アプリケーションに置き換えられます。プレースホルダーアプリケーションは Apache Web Server です。

Amazon ECS の詳細については、「Amazon Elastic Container Service デベロッパーガイド」を参照してください。

プレースホルダーアプリケーションをデプロイするには、以下の一連の手順を実行します。

タスク実行ロールを作成するには

このロールは、ユーザーに代わって API コールを行う AWS Fargate (Fargate) アクセス許可を Amazon ECS に付与します。

1. 信頼ポリシーを作成します。

   a. で AWS CloudShell、次のコマンドを入力します。

```
cat > codecatalyst-ecs-trust-policy.json
```

      CloudShell ターミナルに点滅するプロンプトが表示されます。

   b. プロンプトに次のコードを入力します。

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs-tasks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

c. 最後の中括弧 (}) の後にカーソルを置きます。

d. **Enter** と **Ctrl+d** を押してファイルを保存し、猫を終了します。

2. タスク実行ロールを作成します。

```
aws iam create-role \
      --role-name codecatalyst-ecs-task-execution-role \
      --assume-role-policy-document file://codecatalyst-ecs-trust-policy.json
```

3. AWS 管理AmazonECSTaskExecutionRolePolicyポリシーをロールにアタッチします。

```
aws iam attach-role-policy \
      --role-name codecatalyst-ecs-task-execution-role \
      --policy-arn arn:aws:iam::aws:policy/service-role/
AmazonECSTaskExecutionRolePolicy
```

4. ロールの詳細を表示します。

```
aws iam get-role \
      --role-name codecatalyst-ecs-task-execution-role
```

5. ロール"Arn":の値、例えば を書き留めますarn:aws:iam::111122223333:role/codecatalyst-ecs-task-execution-role。この Amazon リソースネーム (ARN) は後で必要になります。

Amazon ECS クラスターを作成するには

このクラスターには、Apache プレースホルダーアプリケーションと、後で Hello World アプリケーションが含まれます。

1. CodeCatalystECSUserとして、 で空のクラスター AWS CloudShellを作成します。

```
aws ecs create-cluster --cluster-name codecatalyst-ecs-cluster
```

2. (オプション) クラスターが正常に作成されたことを確認します。

```
aws ecs list-clusters
```

codecatalyst-ecs-cluster クラスターの ARN がリストに表示され、正常に作成されたことを示します。

タスク定義ファイルを作成するには

タスク定義ファイルは、 からプルされた Apache 2.4 ウェブサーバーの Docker イメージ (httpd:2.4) を実行することを示します DockerHub。

1. としてCodeCatalystECSUser、 でタスク定義ファイル AWS CloudShellを作成します。

```
cat > taskdef.json
```

2. プロンプトに次のコードを貼り付けます。

```
{
    "executionRoleArn": "arn:aws:iam::111122223333:role/codecatalyst-ecs-task-
execution-role",
    "containerDefinitions": [
        {
            "name": "codecatalyst-ecs-container",
            "image": "httpd:2.4",
            "essential": true,
            "portMappings": [
                {
                    "hostPort": 80,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ]
        }
    ],
    "requiresCompatibilities": [
        "FARGATE"
    ],
```

```
    "cpu": "256",
    "family": "codecatalyst-ecs-task-def",
    "memory": "512",
    "networkMode": "awsvpc"
}
```

上記のコードで、*arn:aws:iam::111122223333:role/codecatalyst-ecs-task-execution-role* を置き換えます。

でメモしたタスク実行ロールの ARN を に含めますタスク実行ロールを作成するには。

3. 最後の中括弧 () の後にカーソルを置きます}。
4. **Enter** と **Ctrl+d**を押してファイルを保存し、猫を終了します。

タスク定義ファイルを Amazon ECS に登録するには

1. としてCodeCatalystECSUser、 にタスク定義 AWS CloudShellを登録します。

```
aws ecs register-task-definition \
    --cli-input-json file://taskdef.json
```

2. (オプション) タスク定義が登録されていることを確認します。

```
aws ecs list-task-definitions
```

codecatalyst-ecs-task-def タスク定義がリストに表示されます。

Amazon ECS サービスを作成するには

Amazon ECS サービスは、Apache プレースホルダーアプリケーションのタスク (および関連する Docker コンテナ) を実行し、後で Hello World アプリケーションを実行します。

1. としてCodeCatalystECSUser、まだ Amazon Elastic Container Service コンソールに切り替えていない場合は、切り替えます。
2. 前に作成したクラスター を選択しますcodecatalyst-ecs-cluster。
3. サービス タブで、 の作成 を選択します。
4. 「作成」ページで、次の操作を行います。

    a. 次にリストされている設定を除き、すべてのデフォルト設定を保持します。

b. [Launch type (起動タイプ)] で、[FARGATE] を選択します。

c. タスク定義 で、ファミリードロップダウンリストで以下を選択します。

```
codecatalyst-ecs-task-def
```

d. サービス名 には、次のように入力します。

```
codecatalyst-ecs-service
```

e. 必要なタスク には、次のように入力します。

```
3
```

このチュートリアルでは、各タスクが単一の Docker コンテナを起動します。

f. ネットワークセクションを展開します。

g. VPC では、任意の VPC を選択します。

h. サブネット で、任意のサブネットを選択します。

> **Note**
>
> サブネットは 1 つだけ指定します。このチュートリアルに必要なのはこれだけです。

> **Note**
>
> VPC とサブネットがない場合は、それらを作成します。「Amazon VPC ユーザーガイド」の「VPC の作成」および「VPC にサブネットを作成する」を参照してください。

i. セキュリティグループ で、新しいセキュリティグループの作成 を選択し、次の操作を行います。

i. セキュリティグループ名 には、次のように入力します。

```
codecatalyst-ecs-security-group
```

ii. セキュリティグループの説明 には、次のように入力します。

```
CodeCatalyst ECS security group
```

iii. [ルールを追加] を選択します。タイプ で HTTP を選択し、ソース で Anywhere を選択します。

j. 下部で、 の作成 を選択します。

k. サービスが作成されるまで待ちます。このプロセスには数分かかることがあります。

5. タスクタブを選択し、更新ボタンを選択します。3 つのタスクすべてで、Last Status 列が Running に設定されていることを確認します。

(オプション) Apache プレースホルダーアプリケーションが実行されていることを確認するには

1. タスク タブで、3 つのタスクのいずれかを選択します。
2. パブリック IP フィールドで、オープンアドレス を選択します。

   `It Works!` ページが表示されます。これは、Amazon ECS サービスが Apache イメージで Docker コンテナを起動するタスクを正常に開始したことを示します。

   チュートリアルのこの時点で、Amazon ECS クラスター、サービス、タスク定義、および Apache プレースホルダーアプリケーションを手動でデプロイしました。これらの項目がすべて揃ったら、Apache プレースホルダーアプリケーションをチュートリアルの Hello World アプリケーションに置き換えるワークフローを作成する準備が整いました。

ステップ 3: Amazon ECR イメージリポジトリを作成する

このセクションでは、Amazon Elastic Container Registry (Amazon ECR) にプライベートイメージリポジトリを作成します。このリポジトリには、以前にデプロイした Apache プレースホルダーイメージを置き換えるチュートリアルの Docker イメージが保存されます。

Amazon ECRの詳細については、Amazon Elastic Container Registry User Guideを参照してください。

Amazon ECR でイメージリポジトリを作成するには

1. としてCodeCatalystECSUser、 で AWS CloudShell Amazon ECR に空のリポジトリを作成します。

```
aws ecr create-repository --repository-name codecatalyst-ecs-image-repo
```

2. Amazon ECR リポジトリの詳細を表示します。

```
aws ecr describe-repositories \
      --repository-names codecatalyst-ecs-image-repo
```

3. “repositoryUri”: 値に注意してください。例えば、 です111122223333.dkr.ecr.us-west-2.amazonaws.com/codecatalyst-ecs-image-repo。

   これは、後でリポジトリをワークフローに追加するときに必要になります。

ステップ 4: AWS ロールを作成する

このセクションでは、 CodeCatalyst ワークフローが機能するために必要な AWS IAM ロールを作成します。これらのロールは次のとおりです。

- ビルドロール – AWS アカウントにアクセスして Amazon ECR と Amazon EC2 に書き込むための CodeCatalyst ビルドアクション (ワークフロー内) アクセス許可を付与します。
- ロールのデプロイ — AWS アカウント CodeCatalyst 、Amazon ECS、およびその他のいくつかのサービスにアクセスするための (ワークフロー内の) ECS アクションへのデプロイアクセス許可を付与します。 AWS

IAM ロールの詳細については、「 ユーザーガイド」の「IAM ロールAWS Identity and Access Management 」を参照してください。

> **Note**
>
> 時間を節約するために、前述の 2 つのロールではなく、CodeCatalystWorkflowDevelopmentRole-*spaceName*ロールと呼ばれる 1 つのロールを作成できます。詳細については、「アカウントとスペースのCodeCatalystWorkflowDevelopmentRole-***spaceName***ロールの作成」を参照してください。CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールには、セキュリティリスクをもたらす可能性のある非常に広範なアクセス許可があることを理解します。このロールは、セキュリティが懸念されないチュートリアルやシナリオでのみ使用することをお勧めします。このチュートリアルでは、前述の 2 つのロールを作成することを前提としています。

ビルドロールとデプロイロールを作成するには、 AWS Management Console または を使用できます AWS CLI。

AWS Management Console

ビルドロールとデプロイロールを作成するには、以下の一連の手順を実行します。

ビルドロールを作成するには

1. 次のように、ロールのポリシーを作成します。
   a. にサインインします AWS。
   b. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
   c. ナビゲーションペインで、ポリシー を選択します。
   d. ポリシーの作成を選択します。
   e. [JSON] タブを選択します。
   f. 既存のコードを削除します。
   g. 次のコードを貼り付けます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:*",
                "ec2:*"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> ロールを初めて使用してワークフローアクションを実行するときは、リソースポリシーステートメントでワイルドカードを使用し、使用可能になった後にリソース名でポリシーの範囲を絞り込みます。

```
        "Resource": "*"
```

h. [次へ : タグ] を選択します。

i. [次へ: レビュー] を選択します。

j. 名前 で、次のように入力します。

```
codecatalyst-ecs-build-policy
```

k. [ポリシーの作成] を選択します。

これで、アクセス許可ポリシーが作成されました。

2. 次のようにビルドロールを作成します。

a. ナビゲーションペインで ロール を選択してから、ロールを作成する を選択します。

b. カスタム信頼ポリシー を選択します。

c. 既存のカスタム信頼ポリシーを削除します。

d. 次のカスタム信頼ポリシーを追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                 ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

e. [次へ] をクリックします。

f. アクセス許可ポリシー で を検索しcodecatalyst-ecs-build-policy、そのチェックボックスをオンにします。

g. [次へ] をクリックします。

h. ロール名 には、次のように入力します。

```
codecatalyst-ecs-build-role
```

i. ロールの説明 には、次のように入力します。

```
CodeCatalyst ECS build role
```

j. [ロールを作成] を選択します。

これで、アクセス許可ポリシーと信頼ポリシーを使用してビルドロールが作成されました。

3. 次のように、ビルドロール ARN を取得します。

a. ナビゲーションペインで、[ロール] を選択します。

b. 検索ボックスに、作成したロールの名前 () を入力しますcodecatalyst-ecs-build-role。

c. リストからロールを選択します。

ロールの概要ページが表示されます。

d. 上部で、ARN 値をコピーします。これは、後で必要になります。

デプロイロールを作成するには

1. 次のように、ロールのポリシーを作成します。

a. にサインインします AWS。

b. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。

c. ナビゲーションペインで、ポリシー を選択します。

d. [Create Policy (ポリシーの作成)] を選択します。

e. [JSON] タブを選択します。

f. 既存のコードを削除します。

g. 次のコードを貼り付けます。

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [{
    "Action":[
      "ecs:DescribeServices",
      "ecs:CreateTaskSet",
      "ecs:DeleteTaskSet",
      "ecs:ListClusters",
      "ecs:RegisterTaskDefinition",
      "ecs:UpdateServicePrimaryTaskSet",
      "ecs:UpdateService",
      "elasticloadbalancing:DescribeTargetGroups",
      "elasticloadbalancing:DescribeListeners",
      "elasticloadbalancing:ModifyListener",
      "elasticloadbalancing:DescribeRules",
      "elasticloadbalancing:ModifyRule",
      "lambda:InvokeFunction",
      "lambda:ListFunctions",
      "cloudwatch:DescribeAlarms",
      "sns:Publish",
      "sns:ListTopics",
      "s3:GetObject",
      "s3:GetObjectVersion",
      "codedeploy:CreateApplication",
      "codedeploy:CreateDeployment",
      "codedeploy:CreateDeploymentGroup",
      "codedeploy:GetApplication",
      "codedeploy:GetDeployment",
      "codedeploy:GetDeploymentGroup",
      "codedeploy:ListApplications",
      "codedeploy:ListDeploymentGroups",
      "codedeploy:ListDeployments",
      "codedeploy:StopDeployment",
      "codedeploy:GetDeploymentTarget",
      "codedeploy:ListDeploymentTargets",
      "codedeploy:GetDeploymentConfig",
      "codedeploy:GetApplicationRevision",
      "codedeploy:RegisterApplicationRevision",
      "codedeploy:BatchGetApplicationRevisions",
      "codedeploy:BatchGetDeploymentGroups",
      "codedeploy:BatchGetDeployments",
      "codedeploy:BatchGetApplications",
      "codedeploy:ListApplicationRevisions",
      "codedeploy:ListDeploymentConfigs",
      "codedeploy:ContinueDeployment"
  ],
```

```
   "Resource":"*",
   "Effect":"Allow"
},{"Action":[
      "iam:PassRole"
   ],
   "Effect":"Allow",
   "Resource":"*",
   "Condition":{"StringLike":{"iam:PassedToService":[
            "ecs-tasks.amazonaws.com",
            "codedeploy.amazonaws.com"
         ]
      }
   }
}]
}
```

> **Note**
>
> ロールを初めて使用してワークフローアクションを実行する場合は、リソースポリシーステートメントでワイルドカードを使用します。その後、リソース名が使用可能になったら、そのリソース名を使用してポリシーの範囲を絞り込むことができます。
>
> ```
> "Resource": "*"
> ```

h. [次へ : タグ] を選択します。

i. [次へ: レビュー] を選択します。

j. 名前 で、次のように入力します。

```
codecatalyst-ecs-deploy-policy
```

k. [ポリシーの作成] を選択します。

これで、アクセス許可ポリシーが作成されました。

2. デプロイロールを次のように作成します。

a. ナビゲーションペインで ロール を選択してから、ロールを作成する を選択します。

b. カスタム信頼ポリシー を選択します。

c. 既存のカスタム信頼ポリシーを削除します。

d. 次のカスタム信頼ポリシーを追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

e. [次へ] をクリックします。

f. アクセス許可ポリシー で を検索しcodecatalyst-ecs-deploy-policy、そのチェックボックスをオンにします。

g. [次へ] をクリックします。

h. ロール名 には、次のように入力します。

```
codecatalyst-ecs-deploy-role
```

i. ロールの説明 には、次のように入力します。

```
CodeCatalyst ECS deploy role
```

j. [ロールを作成] を選択します。

これで、信頼ポリシーを持つデプロイロールが作成されました。

3. 次のように、デプロイロール ARN を取得します。

a. ナビゲーションペインで、[ロール] を選択します。

b. 検索ボックスに、作成したロールの名前 () を入力しますcodecatalyst-ecs-deploy-role。

c. リストからロールを選択します。

ロールの概要ページが表示されます。

d. 上部で、ARN 値をコピーします。これは、後で必要になります。

AWS CLI

ビルドロールとデプロイロールを作成するには、以下の一連の手順を実行します。

両方のロールの信頼ポリシーを作成するには

としてCodeCatalystECSUser、 で信頼ポリシーファイル AWS CloudShellを作成します。

1. ファイルを作成します。

```
cat > codecatalyst-ecs-trust-policy.json
```

2. ターミナルプロンプトで、次のコードを貼り付けます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

3. 最後の中括弧 (}) の後にカーソルを置きます。

4. **Enter** と **Ctrl+d**を押してファイルを保存し、猫を終了します。

ビルドポリシーとビルドロールを作成するには

1. ビルドポリシーを作成します。

   a. としてCodeCatalystECSUser、 でビルドポリシーファイル AWS CloudShellを作成します。

   ```
   cat > codecatalyst-ecs-build-policy.json
   ```

   b. プロンプトで、次のコードを入力します。

   ```
   {
       "Version": "2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": [
                   "ecr:*",
                   "ec2:*"
               ],
               "Resource": "*"
           }
       ]
   }
   ```

   c. 最後の中括弧 (}) の後にカーソルを置きます。

   d. **Enter** と **Ctrl+d**を押してファイルを保存し、猫を終了します。

2. ビルドポリシーを に追加します AWS。

   ```
   aws iam create-policy \
       --policy-name codecatalyst-ecs-build-policy \
       --policy-document file://codecatalyst-ecs-build-policy.json
   ```

3. コマンド出力で、 などの"arn":値を書き留めますarn:aws:iam::111122223333:policy/codecatalyst-ecs-build-policy。このARN は後で必要になります。

4. ビルドロールを作成し、信頼ポリシーをアタッチします。

   ```
   aws iam create-role \
         --role-name codecatalyst-ecs-build-role \
   ```

```
      --assume-role-policy-document file://codecatalyst-ecs-trust-policy.json
```

5. ビルドポリシーをビルドロールにアタッチします。

```
aws iam attach-role-policy \
      --role-name codecatalyst-ecs-build-role \
      --policy-arn arn:aws:iam::111122223333:policy/codecatalyst-ecs-build-
policy
```

ここで*#arn:aws:iam::111122223333:policy/codecatalyst-ecs-build-policy* は、前にメモしたビルドポリシーの ARN に置き換えられます。

6. ビルドロールの詳細を表示します。

```
aws iam get-role \
      --role-name codecatalyst-ecs-build-role
```

7. ロール"Arn":の値、例えば を書き留めますarn:aws:iam::111122223333:role/codecatalyst-ecs-build-role。この ARN は後で必要になります。

デプロイポリシーとデプロイロールを作成するには

1. デプロイポリシーを作成します。

   a. で AWS CloudShell、デプロイポリシーファイルを作成します。

   ```
   cat > codecatalyst-ecs-deploy-policy.json
   ```

   b. プロンプトで、次のコードを入力します。

   ```
   {
       "Version": "2012-10-17",
       "Statement": [{
       "Action":[
         "ecs:DescribeServices",
         "ecs:CreateTaskSet",
         "ecs:DeleteTaskSet",
         "ecs:ListClusters",
         "ecs:RegisterTaskDefinition",
         "ecs:UpdateServicePrimaryTaskSet",
         "ecs:UpdateService",
         "elasticloadbalancing:DescribeTargetGroups",
   ```

```
      "elasticloadbalancing:DescribeListeners",
      "elasticloadbalancing:ModifyListener",
      "elasticloadbalancing:DescribeRules",
      "elasticloadbalancing:ModifyRule",
      "lambda:InvokeFunction",
      "lambda:ListFunctions",
      "cloudwatch:DescribeAlarms",
      "sns:Publish",
      "sns:ListTopics",
      "s3:GetObject",
      "s3:GetObjectVersion",
      "codedeploy:CreateApplication",
      "codedeploy:CreateDeployment",
      "codedeploy:CreateDeploymentGroup",
      "codedeploy:GetApplication",
      "codedeploy:GetDeployment",
      "codedeploy:GetDeploymentGroup",
      "codedeploy:ListApplications",
      "codedeploy:ListDeploymentGroups",
      "codedeploy:ListDeployments",
      "codedeploy:StopDeployment",
      "codedeploy:GetDeploymentTarget",
      "codedeploy:ListDeploymentTargets",
      "codedeploy:GetDeploymentConfig",
      "codedeploy:GetApplicationRevision",
      "codedeploy:RegisterApplicationRevision",
      "codedeploy:BatchGetApplicationRevisions",
      "codedeploy:BatchGetDeploymentGroups",
      "codedeploy:BatchGetDeployments",
      "codedeploy:BatchGetApplications",
      "codedeploy:ListApplicationRevisions",
      "codedeploy:ListDeploymentConfigs",
      "codedeploy:ContinueDeployment"
   ],
   "Resource":"*",
   "Effect":"Allow"
},{"Action":[
      "iam:PassRole"
   ],
   "Effect":"Allow",
   "Resource":"*",
   "Condition":{"StringLike":{"iam:PassedToService":[
            "ecs-tasks.amazonaws.com",
            "codedeploy.amazonaws.com"
```

```
            ]
        }
    }
}]
}
```

> **Note**
>
> ロールを初めて使用してワークフローアクションを実行するときは、リソースポリシーステートメントでワイルドカードを使用し、使用可能になった後にリソース名でポリシーの範囲を絞り込みます。
>
> ```
> "Resource": "*"
> ```

c. 最後の中括弧 () の後にカーソルを置きます}。

d. **Enter** と **Ctrl+d**を押してファイルを保存し、猫を終了します。

2. デプロイポリシーを に追加します AWS。

```
aws iam create-policy \
    --policy-name codecatalyst-ecs-deploy-policy \
    --policy-document file://codecatalyst-ecs-deploy-policy.json
```

3. コマンド出力で、デプロイポリシーの値"arn":、例えば を書き留めますarn:aws:iam::111122223333:policy/codecatalyst-ecs-deploy-policy。この ARN は後で必要になります。

4. デプロイロールを作成し、信頼ポリシーをアタッチします。

```
aws iam create-role \
      --role-name codecatalyst-ecs-deploy-role \
      --assume-role-policy-document file://codecatalyst-ecs-trust-policy.json
```

5. デプロイポリシーをデプロイロールにアタッチします。ここで、*arn:aws:iam::111122223333:policy/codecatalyst-ecs-deploy-policy* は、前にメモしたデプロイポリシーの ARN に置き換えられます。

```
aws iam attach-role-policy \
      --role-name codecatalyst-ecs-deploy-role \
```

```
      --policy-arn arn:aws:iam::111122223333:policy/codecatalyst-ecs-deploy-
policy
```

6. デプロイロールの詳細を表示します。

```
aws iam get-role \
      --role-name codecatalyst-ecs-deploy-role
```

7. ロールの値"Arn":、例えば を書き留めますarn:aws:iam::111122223333:role/codecatalyst-ecs-deploy-role。この ARN は後で必要になります。

ステップ 5: に AWS ロールを追加する CodeCatalyst

このステップでは、ビルドロール (codecatalyst-ecs-build-role) とデプロイロール (codecatalyst-ecs-deploy-role) をスペース内の CodeCatalyst アカウント接続に追加します。

ビルドロールとデプロイロールをアカウント接続に追加するには

1. で CodeCatalyst、スペースに移動します。
2. [AWS アカウント] を選択します。アカウント接続のリストが表示されます。
3. ビルドロールとデプロイロールを作成したアカウントを表す AWS アカウント接続を選択します。
4. AWS 管理コンソール からロールの管理 を選択します。

   「Amazon CodeCatalyst スペースへの IAM ロールの追加」ページが表示されます。ページにアクセスするには、サインインが必要な場合があります。
5. IAM で作成した既存のロールを追加する を選択します。

   ドロップダウンリストが表示されます。このリストには、 codecatalyst-runner.amazonaws.comおよび codecatalyst.amazonaws.comサービスプリンシパルを含む信頼ポリシーを持つすべての IAM ロールが表示されます。
6. ドロップダウンリストで、 を選択しcodecatalyst-ecs-build-role、ロールの追加 を選択します。

> **Note**
>
> が表示された場合はThe security token included in the request is invalid、適切なアクセス許可がないためである可能性があります。この問題を解決す

るには、 からサインアウト AWS して、スペースの作成時に使用した AWS アカウントで再度サインインします CodeCatalyst。

7. 「IAM ロールの追加」を選択し、「IAM で作成した既存のロールの追加」を選択し、ドロップダウンリストから「」を選択しますcodecatalyst-ecs-deploy-role。[Add role] を選択します。

   これで、ビルドロールとデプロイロールをスペースに追加しました。

8. Amazon CodeCatalyst 表示名 の値をコピーします。この値は、後でワークフローを作成するときに必要になります。

ステップ 6: ソースリポジトリを作成する

このステップでは、 でソースリポジトリを作成します CodeCatalyst。このリポジトリには、タスク定義ファイルなど、チュートリアルのソースファイルが保存されます。

ソースリポジトリの詳細については、「」を参照してくださいソースリポジトリの作成。

ソースリポジトリを作成するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクト に移動しますcodecatalyst-ecs-project。
3. ナビゲーションペインで [コード] を選択してから、[ソースリポジトリ] を選択します。
4. [リポジトリの追加] を選択し、[リポジトリの作成] を選択します。
5. リポジトリ名 で、次のように入力します。

   ```
   codecatalyst-ecs-source-repository
   ```

6. [作成] を選択します。

ステップ 7: ソースファイルを追加する

このセクションでは、Hello World ソースファイルを CodeCatalyst リポジトリ に追加しますcodecatalyst-ecs-source-repository。これらは以下で構成されます。

- index.html ファイル - ブラウザに Hello World メッセージを表示します。
- Dockerfile – Docker イメージに使用するベースイメージと、それに適用する Docker コマンドについて説明します。

- taskdef.json ファイル - クラスターでタスクを起動するときに使用する Docker イメージを定義します。

フォルダ構造は次のとおりです。

```
.
|– public-html
|   |– index.html
|– Dockerfile
|– taskdef.json
```

> **Note**
>
> 次の手順は、コンソールを使用してファイルを追加する方法を示しています CodeCatalyst が、必要に応じて Git を使用できます。詳細については、「ソースリポジトリのクローン作成」を参照してください。

トピック

- index.html
- Dockerfile
- taskdef.json

index.html

index.html ファイルには、ブラウザに Hello World メッセージが表示されます。

index.html ファイルを追加するには

1. CodeCatalyst コンソールで、ソースリポジトリ に移動しますcodecatalyst-ecs-source-repository。
2. ファイル で、ファイルの作成 を選択します。
3. ファイル名 には、次のように入力します。

   public-html/index.html

> **Important**
>
> 同じ名前のフォルダを作成するには、必ず public-html/ プレフィックスを含めてください。index.html はこのフォルダにあることが想定されます。

4. テキストボックスに、次のコードを入力します。

```
<html>
  <head>
    <title>Hello World</title>
    <style>
      body {
      background-color: black;
      text-align: center;
      color: white;
      font-family: Arial, Helvetica, sans-serif;
      }
    </style>
  </head>
  <body>
    <h1>Hello World</h1>
  </body>
</html>
```

5. コミット を選択し、もう一度コミット を選択します。

   index.html は、 public-htmlフォルダのリポジトリに追加されます。

Dockerfile

Dockerfile は、使用するベース Docker イメージと、それに適用する Docker コマンドを記述します。Dockerfile の詳細については、「Dockerfile リファレンス」を参照してください。

ここで指定された Dockerfile は、Apache 2.4 ベースイメージ (httpd) を使用することを示します。また、ウェブページを提供する Apache サーバーのフォルダindex.htmlに というソースファイルをコピーする手順も含まれています。Dockerfile の EXPOSE命令は、コンテナがポート 80 でリッスンしていることを Docker に伝えます。

Dockerfile を追加するには

1. ソースリポジトリで、ファイルの作成 を選択します。
2. ファイル名 には、次のように入力します。

   ```
   Dockerfile
   ```

   ファイル拡張子を含めないでください。

   > **Important**
   >
   > Dockerfile はリポジトリのルートフォルダに存在する必要があります。ワークフローのDocker buildコマンドは、ワークフローが存在することを期待します。

3. テキストボックスに、次のコードを入力します。

   ```
   FROM httpd:2.4
   COPY ./public-html/index.html /usr/local/apache2/htdocs/index.html
   EXPOSE 80
   ```

4. コミット を選択し、もう一度コミット を選択します。

   Dockerfile がリポジトリに追加されます。

taskdef.json

このステップで追加するファイルは、 で既に指定したtaskdef.jsonファイルと同じですが、ステップ 2: プレースホルダーアプリケーションを Amazon ECS にデプロイする次の違いがあります。

ここでのタスク定義では、image:フィールド (httpd:2.4) でハードコードされた Docker イメージ名を指定する代わりに、いくつかの変数を使用してイメージ $REPOSITORY_URIおよび を示します$IMAGE_TAG。これらの変数は、後のステップでワークフローを実行するときに、ワークフローのビルドアクションによって生成された実際の値に置き換えられます。

タスク定義パラメータの詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「タスク定義パラメータ」を参照してください。

taskdef.json ファイルを追加するには

1. ソースリポジトリで、ファイルの作成 を選択します。

2. ファイル名 には、次のように入力します。

```
taskdef.json
```

3. テキストボックスに、次のコードを入力します。

```
{
    "executionRoleArn": "arn:aws:iam::account_ID:role/codecatalyst-ecs-task-
execution-role",
    "containerDefinitions": [
        {
            "name": "codecatalyst-ecs-container",
            # The $REPOSITORY_URI and $IMAGE_TAG variables will be replaced
            # by the workflow at build time (see the build action in the
            # workflow)
            "image": $REPOSITORY_URI:$IMAGE_TAG,
            "essential": true,
            "portMappings": [
                {
                    "hostPort": 80,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ]
        }
    ],
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "networkMode": "awsvpc",
    "cpu": "256",
    "memory": "512",
    "family": "codecatalyst-ecs-task-def"
}
```

上記のコードで、

arn:aws:iam::account_ID:role/codecatalyst-ecs-task-execution-role

でメモしたタスク実行ロールの ARN を持つ タスク実行ロールを作成するには。

4. コミット を選択し、もう一度コミット を選択します。

taskdef.json ファイルがリポジトリに追加されます。

ステップ 8: ワークフローを作成して実行する

このステップでは、ソースファイルを取得し、Docker イメージにビルドして、そのイメージを Amazon ECS クラスターにデプロイするワークフローを作成します。このデプロイは、既存の Apache プレースホルダーアプリケーションを置き換えます。

ワークフローは、順番に実行される次の構成要素で構成されます。

- トリガー — このトリガーは、変更をソースリポジトリにプッシュすると、ワークフローの実行を自動的に開始します。トリガーについての詳細は、「トリガーを使用したワークフローの自動実行の開始」を参照してください。
- ビルドアクション (BuildBackend) – トリガー時に、 アクションは Dockerfile を使用して Docker イメージをビルドし、そのイメージを Amazon ECR にプッシュします。ビルドアクションは、 を正しいimageフィールド値taskdef.jsonで更新し、このファイルの出力アーティファクトを作成します。このアーティファクトは、デプロイアクションの入力として使用され、次は です。

  ビルドアクションの詳細については、「」を参照してくださいワークフローによる構築。
- デプロイアクション (DeployToECS) – ビルドアクションが完了すると、デプロイアクションはビルドアクション (TaskDefArtifact) によって生成された出力アーティファクトを検索し、そのtaskdef.json内部を見つけて、Amazon ECS サービスに登録します。次に、サービスは taskdef.json ファイルの指示に従って、Amazon ECS クラスター内で 3 つの Amazon ECS タスク、および関連付けられた Hello World Docker コンテナを実行します。

ワークフローを作成するには

1. CodeCatalyst コンソールのナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
2. ワークフローの作成 を選択します。
3. ソースリポジトリ で、 を選択しますcodecatalyst-ecs-source-repository。
4. ブランチ で、 を選択しますmain。
5. [作成] を選択します。
6. YAML サンプルコードを削除します。
7. 次の YAML コードを追加します。

```
Name: codecatalyst-ecs-workflow
SchemaVersion: 1.0

Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  BuildBackend:
    Identifier: aws/build@v1
    Environment:
      Name: codecatalyst-ecs-environment
      Connections:
        - Name: codecatalyst-account-connection
          Role: codecatalyst-ecs-build-role
    Inputs:
      Sources:
        - WorkflowSource
      Variables:
        - Name: REPOSITORY_URI
          Value: 111122223333.dkr.ecr.us-west-2.amazonaws.com/codecatalyst-ecs-
image-repo
        - Name: IMAGE_TAG
          Value: ${WorkflowSource.CommitId}
    Configuration:
      Steps:
        #pre_build:
        - Run: echo Logging in to Amazon ECR...
        - Run: aws --version
        - Run: aws ecr get-login-password --region us-west-2 | docker login --
username AWS --password-stdin 111122223333.dkr.ecr.us-west-2.amazonaws.com
        #build:
        - Run: echo Build started on `date`
        - Run: echo Building the Docker image...
        - Run: docker build -t $REPOSITORY_URI:latest .
        - Run: docker tag $REPOSITORY_URI:latest $REPOSITORY_URI:$IMAGE_TAG
        #post_build:
        - Run: echo Build completed on `date`
        - Run: echo Pushing the Docker images...
        - Run: docker push $REPOSITORY_URI:latest
        - Run: docker push $REPOSITORY_URI:$IMAGE_TAG
        # Replace the variables in taskdef.json
```

```
        - Run: find taskdef.json -type f | xargs sed -i "s|\$REPOSITORY_URI|
$REPOSITORY_URI|g"
        - Run: find taskdef.json -type f | xargs sed -i "s|\$IMAGE_TAG|$IMAGE_TAG|
g"
        - Run: cat taskdef.json
        # The output artifact will be a zip file that contains a task definition
 file.
    Outputs:
      Artifacts:
        - Name: TaskDefArtifact
          Files:
            - taskdef.json
  DeployToECS:
    DependsOn:
      - BuildBackend
    Identifier: aws/ecs-deploy@v1
    Environment:
      Name: codecatalyst-ecs-environment
      Connections:
        - Name: codecatalyst-account-connection
          Role: codecatalyst-ecs-deploy-role
    Inputs:
      Sources: []
      Artifacts:
        - TaskDefArtifact
    Configuration:
      region: us-west-2
      cluster: codecatalyst-ecs-cluster
      service: codecatalyst-ecs-service
      task-definition: taskdef.json
```

上記のコードで、以下を置き換えます。

- で作成した環境の名前*codecatalyst-ecs-environment*を持つ の両方のインスタンス前提条件。
- アカウント接続の表示名*codecatalyst-account-connection*を持つ の両方のインスタンス。表示名は数字にすることができます。詳細については、「ステップ 5: に AWS ロールを追加する CodeCatalyst」を参照してください。
- *codecatalyst-ecs-build-role* で作成したビルドロールの名前。 ステップ 4: AWS ロールを作成する

- *111122223333.dkr.ecr.us-west-2.amazonaws.com/codecatalyst-ecs-image-repo* (Value:プロパティ内) と、 で作成した Amazon ECR リポジトリの URIステップ 3: Amazon ECR イメージリポジトリを作成する。
- イメージサフィックス () のない Amazon ECR リポジトリの URI を持つ *111122223333.dkr.ecr.us-west-2.amazonaws.com* (Run: aws ecr コマンド内/ codecatalyst-ecs-image-repo) 。
- *codecatalyst-ecs-deploy-role* で作成したデプロイロールの名前。 ステップ 4: AWS ロールを作成する
- us-*west-2* のインスタンスは両方とも AWS 、リージョンコードで使用します。リージョンコードのリストについては、「」の「リージョンエンドポイント」を参照してくださいAWS 全般のリファレンス。

> **Note**
>
> ビルドロールとデプロイロールを作成しない場合は、 *codecatalyst-ecs-build-role* と をCodeCatalystWorkflowDevelopmentRole-*spaceName*ロールの名前*codecatalyst-ecs-deploy-role*に置き換えます。このロールの詳細については、「ステップ 4: AWS ロールを作成する」を参照してください。

> **Tip**
>
> 前のワークフローコードに示されている find および sed コマンドを使用してリポジトリとイメージ名を更新する代わりに、この目的のために Amazon ECS タスク定義アクションをレンダリングできます。詳細については、「ワークフローを使用した Amazon ECS タスク定義ファイルの変更」を参照してください。

8. (オプション) 検証 を選択して、コミットする前に YAML コードが有効であることを確認します。
9. [Commit] (コミット) を選択します。
10. コミットワークフローダイアログボックスに、次のように入力します。

    a. コミットメッセージ で、テキストを削除して次のように入力します。

```
Add first workflow
```

b. リポジトリ で、 を選択しますcodecatalyst-ecs-source-repository。

c. ブランチ名 で、main を選択します。

d. [Commit] (コミット) を選択します。

これでワークフローが作成されました。ワークフロー実行は、ワークフローの上部で定義されたトリガーが原因で自動的に開始されます。具体的には、workflow.yamlファイルをソースリポジトリにコミット (およびプッシュ) すると、トリガーによってワークフロー実行が開始されます。

ワークフロー実行の進行状況を表示するには

1. CodeCatalyst コンソールのナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
2. 先ほど作成したワークフロー を選択しますcodecatalyst-ecs-workflow。
3. ビルドの進行状況BuildBackendを表示するには、 を選択します。
4. DeployToECS を選択すると、デプロイの進行状況が表示されます。

   実行の詳細の表示の詳細については、「」を参照してくださいワークフローの実行ステータスと詳細の表示。

デプロイを確認するには

1. Amazon ECS クラシックコンソール (https://console.aws.amazon.com/ecs/) を開きます。
2. クラスター を選択しますcodecatalyst-ecs-cluster。
3. [タスク] タブを選択します。
4. 3 つのタスクのいずれかを選択します。
5. パブリック IP フィールドで、オープンアドレス を選択します。

   ブラウザに「Hello World」ページが表示され、Amazon ECS サービスがアプリケーションを正常にデプロイしたことを示します。

ステップ 9: ソースファイルを変更する

このセクションでは、ソースリポジトリの index.html ファイルを変更します。この変更により、ワークフローは新しい Docker イメージを構築し、コミット ID でタグ付けして Amazon ECR にプッシュし、Amazon ECS にデプロイします。

index.html を変更するには

1. CodeCatalyst コンソールのナビゲーションペインで、コード を選択し、ソースリポジトリ を選択し、リポジトリ を選択しますcodecatalyst-ecs-source-repository。
2. [public-html] を選択し、[index.html] を選択します。

   の内容index.htmlが表示されます。
3. [編集] を選択します。
4. 14 行目で、Hello Worldテキストを に変更しますTutorial complete!。
5. コミット を選択し、もう一度コミット を選択します。

   コミットにより、新しいワークフロー実行が開始されます。
6. (オプション) ソースリポジトリのメインページに移動し、コミットの表示 を選択し、index.html変更のコミット ID を書き留めます。
7. デプロイの進行状況を確認します。
   a. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
   b. を選択して最新の実行codecatalyst-ecs-workflowを表示します。
   c. ワークフローの実行の進行状況を確認するにはBuildBackend、、、および DeployToECS を選択します。
8. 次のように、アプリケーションが更新されていることを確認します。
   a. Amazon ECS クラシックコンソール (https://console.aws.amazon.com/ecs/) を開きます。
   b. クラスター を選択しますcodecatalyst-ecs-cluster。
   c. [タスク] タブを選択します。
   d. 3 つのタスクのいずれかを選択します。
   e. パブリック IP フィールドで、オープンアドレス を選択します。

      Tutorial complete! ページが表示されます。

9. （オプション) で AWS Amazon ECR コンソールに切り替え、新しい Docker イメージにステップ 6 のコミット ID がタグ付けされていることを確認します。

クリーンアップ

このチュートリアルで使用するファイルとサービスをクリーンアップして、料金が発生しないようにします。

で AWS Management Console、次の順序でクリーンアップします。

1. Amazon ECS で、次の操作を行います。
   a. を削除しますcodecatalyst-ecs-service。
   b. を削除しますcodecatalyst-ecs-cluster。
   c. codecatalyst-ecs-task-definition の登録を解除します。
2. Amazon ECR で、 を削除しますcodecatalyst-ecs-image-repo。
3. Amazon EC2 で、 を削除しますcodecatalyst-ecs-security-group。
4. IAM Identity Center で、以下を削除します。
   a. CodeCatalystECSUser
   b. CodeCatalystECSPermissionSet

CodeCatalyst コンソールで、次のようにクリーンアップします。

1. を削除しますcodecatalyst-ecs-workflow。
2. を削除しますcodecatalyst-ecs-environment。
3. を削除しますcodecatalyst-ecs-source-repository。
4. を削除しますcodecatalyst-ecs-project。

このチュートリアルでは、 CodeCatalyst ワークフローと Amazon ECS へのデプロイアクションを使用して Amazon ECS サービスにアプリケーションをデプロイする方法を学習しました。

「Amazon ECS へのデプロイ」アクションの追加

次の手順を使用して、Amazon ECS へのデプロイアクションをワークフローに追加します。

Visual

ビジュアルエディタを使用して「Amazon ECS へのデプロイ」アクションを追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. ビジュアル を選択します。
7. 左上で、+ Actions を選択してアクションカタログを開きます。
8. ドロップダウンリストから Amazon CodeCatalystを選択します。
9. Amazon ECS へのデプロイアクションを検索し、次のいずれかを実行します。
   - プラス記号 (+) を選択してワークフロー図にアクションを追加し、設定ペインを開きます。

     または

   - Amazon ECS にデプロイを選択します。アクションの詳細ダイアログボックスが表示されます。このダイアログボックスで、次の操作を行います。
     - (オプション) ダウンロード を選択して、アクションのソースコード を表示します。
     - ワークフローに追加 を選択して、ワークフロー図にアクションを追加し、その設定ペインを開きます。
10. 入力タブと設定タブで、必要に応じてフィールドに入力します。各フィールドの説明については、「」を参照してください「Amazon ECS にデプロイ」アクション YAML 定義。このリファレンスでは、YAML エディタとビジュアルエディタの両方に表示される各フィールド (および対応する YAML プロパティ値) に関する詳細情報を提供します。
11. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
12. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

YAML

YAML エディタを使用して「Amazon ECS へのデプロイ」アクションを追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. YAML を選択します。
7. 左上で + Actions を選択してアクションカタログを開きます。
8. ドロップダウンリストから Amazon CodeCatalystを選択します。
9. Amazon ECS へのデプロイアクションを検索し、次のいずれかを実行します。

   - プラス記号 (+) を選択してワークフロー図にアクションを追加し、その設定ペインを開きます。

     または

   - Amazon ECS にデプロイを選択します。アクションの詳細ダイアログボックスが表示されます。このダイアログボックスで、次の操作を行います。
     - (オプション) ダウンロード を選択して、アクションのソースコード を表示します。
     - ワークフローに追加 を選択して、ワークフロー図にアクションを追加し、その設定ペインを開きます。
10. 必要に応じて YAML コードのプロパティを変更します。使用可能な各プロパティの説明は、「」に記載されています「Amazon ECS にデプロイ」アクション YAML 定義。
11. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
12. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

「Amazon ECS へのデプロイ」アクションによって生成される変数

Amazon ECS にデプロイ アクションは、実行時に次の変数を生成して設定します。これらは事前定義された変数 と呼ばれます。

ワークフローでこれらの変数を参照する方法については、「」を参照してください事前定義された変数の使用。

キー	値
クラスター	ワークフローの実行中にデプロイされた Amazon ECS クラスターの名前。 例: `codecatalyst-ecs-cluster`
デプロイプラットフォーム	デプロイプラットフォームの名前。 にハードコードされていますAWS:ECS。
service	ワークフローの実行中に にデプロイされた Amazon ECS サービスの名前。 例: `codecatalyst-ecs-service`
task-definition-arn	ワークフローの実行中に登録されたタスク定義の Amazon リソースネーム (ARN)。 例: `arn:aws:ecs:us-west-2:111122223333:task-definition/codecatalyst-task-def:8` 前の例:8の は、登録されたリビジョンを示しています。
デプロイ URL	Amazon ECS コンソールのイベントタブへのリンク。ワークフロー実行に関連付けられた Amazon ECS デプロイの詳細を表示できます。 例: `https://console.aws.amazon.com/ecs/home?region=us-west-2#/clusters/codecatalyst-ecs-cluster/services/codecatalyst-ecs-service/events`

キー	値
region	ワークフローの実行中に にデプロイ AWS リージョン された のリージョンコード。 例: us-west-2

「Amazon ECS にデプロイ」アクション YAML 定義

Amazon ECS へのデプロイアクションの YAML 定義を次に示します。このアクションの使用方法については、「」を参照してくださいワークフローを使用した Amazon Elastic Container Service (ECS) へのアプリケーションのデプロイ。

このアクション定義は、より広範なワークフロー定義ファイル内のセクションとして存在します。ファイルの詳細については、「ワークフロー YAML 定義」を参照してください。

> **Note**
>
> 後続の YAML プロパティのほとんどには、対応する UI 要素がビジュアルエディタにあります。UI 要素を検索するには、Ctrl+F を使用します。 要素は、関連付けられた YAML プロパティとともに一覧表示されます。

```
# The workflow definition starts here.
# See ######### for details.

Name: MyWorkflow
SchemaVersion: 1.0
Actions:

# The action definition starts here.
  ECSDeployAction_nn:
    Identifier: aws/ecs-deploy@v1
    DependsOn:
      - build-action
    Compute:
      Type: EC2 | Lambda
      Fleet: fleet-name
    Timeout: timeout-minutes
    Environment:
```

```
      Name: environment-name
      Connections:
        - Name: account-connection-name
          Role: DeployToECS
    Inputs:
      # Specify a source or an artifact, but not both.
      Sources:
        - source-name-1
      Artifacts:
        - task-definition-artifact
    Configuration:
      region: us-east-1
      cluster: ecs-cluster
      service: ecs-service
      task-definition: task-definition-path
      force-new-deployment: false|true
      codedeploy-appspec: app-spec-file-path
      codedeploy-application: application-name
      codedeploy-deployment-group: deployment-group-name
      codedeploy-deployment-description: deployment-description
```

ECSDeployAction

(必須)

アクションの名前を指定します。すべてのアクション名は、ワークフロー内で一意である必要があります。アクション名は、英数字 (a～z、A～Z、0～9)、ハイフン (-)、アンダースコア (_) に制限されています。スペースは使用できません。引用符を使用してアクション名で特殊文字やスペースを有効にすることはできません。

デフォルト: ECSDeployAction_nn。

対応する UI: 設定タブ/アクションの表示名

Identifier

(*ECSDeployAction*/Identifier)

(必須)

アクションを識別します。バージョンを変更しない限り、このプロパティを変更しないでください。詳細については、「アクションのメジャー、マイナー、またはパッチバージョンの指定」を参照してください。

デフォルト: `aws/ecs-deploy@v1`。

対応する UI: ワークフロー図/ECS DeployAction_nn/aws/ecs-deploy@v1 ラベル

DependsOn

(*ECSDeployAction*/DependsOn)

(オプション)

このアクションを実行するために正常に実行する必要があるアクション、アクショングループ、またはゲートを指定します。

「依存」機能の詳細については、「」を参照してください他のアクションに依存するようにアクションを設定する。

対応する UI: の入力タブ/依存 - オプション

Compute

(*ECSDeployAction*/Compute)

(オプション)

ワークフローアクションを実行するために使用されるコンピューティングエンジン。コンピューティングはワークフローレベルまたはアクションレベルで指定できますが、両方を指定することはできません。ワークフローレベルで指定すると、コンピューティング設定はワークフローで定義されたすべてのアクションに適用されます。ワークフローレベルでは、同じインスタンスで複数のアクションを実行することもできます。詳細については、「アクション間でのコンピューティングの共有」を参照してください。

対応する UI: なし

Type

(*ECSDeployAction*/Compute/Type)

(Computeが含まれている場合は必須)

コンピューティングエンジンのタイプ。次のいずれかの値を使用できます。

- EC2 (ビジュアルエディタ) または EC2 (YAML エディタ)

  アクション実行中の柔軟性のために最適化されました。

- Lambda (ビジュアルエディタ) または Lambda (YAML エディタ)

  アクションの起動速度を最適化しました。

コンピューティングタイプの詳細については、「コンピューティングタイプ」を参照してください。

対応する UI: 設定タブ/詳細 - オプション/コンピューティングタイプ

Fleet

(*ECSDeployAction*/Compute/Fleet)

(オプション)

ワークフローまたはワークフローアクションを実行するマシンまたはフリートを指定します。オンデマンドフリートでは、アクションが開始されると、ワークフローは必要なリソースをプロビジョニングし、アクションが終了するとマシンは破棄されます。オンデマンドフリートの例: `Linux.x86-64.Large`、`Linux.x86-64.XLarge`。オンデマンドフリートの詳細については、「」を参照してくださいオンデマンドフリートのプロパティ。

プロビジョニングされたフリートでは、ワークフローアクションを実行するように一連の専用マシンを設定します。これらのマシンはアイドル状態のままで、すぐにアクションを処理できます。プロビジョニングされたフリートの詳細については、「」を参照してくださいプロビジョニングされたフリートのプロパティ。

`Fleet` を省略した場合、デフォルトは ですLinux.x86-64.Large。

対応する UI: 設定タブ/詳細 - オプション/コンピューティングフリート

Timeout

(*ECSDeployAction*/Timeout)

(オプション)

がアクション CodeCatalyst を終了するまでにアクションを実行できる時間を分単位で指定します (YAML エディタ)、または時間と分単位で指定します (ビジュアルエディタ)。最小値は 5 分で、最大値は「」で説明されていますワークフローのクォータ。デフォルトのタイムアウトは、最大タイムアウトと同じです。

対応する UI: 設定タブ/タイムアウト - オプション

Environment

(*ECSDeployAction*/Environment)

(必須)

アクションで使用する CodeCatalyst 環境を指定します。

環境の詳細については、環境を使用して AWS アカウント および VPCsにデプロイする CodeCatalyst「」および「」を参照してください環境を作成する。

対応する UI: 設定タブ/'環境/アカウント/ロール'/環境

Name

(*ECSDeployAction*/Environment/Name)

(Environmentが含まれている場合は必須)

アクションに関連付ける既存の環境の名前を指定します。

対応する UI: 設定タブ/'環境/アカウント/ロール'/環境

Connections

(*ECSDeployAction*/Environment/Connections)

(Environmentが含まれている場合は必須)

アクションに関連付けるアカウント接続を指定します。で最大 1 つのアカウント接続を指定できますEnvironment。

アカウント接続の詳細については、「」を参照してください接続された AWS リソースへのアクセスを許可する AWS アカウント。アカウント接続を環境に関連付ける方法については、「」を参照してください環境を作成する。

対応する UI: 設定タブ/'環境/アカウント/ロール'/AWS アカウント接続

Name

(*ECSDeployAction*/Environment/Connections/Name)

(必須)

アカウント接続の名前を指定します。

対応する UI: 設定タブ/'環境/アカウント/ロール'/AWS アカウント接続

Role

(*ECSDeployAction*/Environment/Connections/Role)

(必須)

Amazon ECS へのデプロイアクションが にアクセスするために使用する IAM ロールの名前を指定します AWS。このロールに次のポリシーが含まれていることを確認します。

- 次のアクセス許可ポリシー :

> **Warning**
>
> アクセス許可を次のポリシーに示すものに制限します。より広範なアクセス許可を持つロールを使用すると、セキュリティ上のリスクが生じる可能性があります。

```
{
    "Version": "2012-10-17",
    "Statement": [{
    "Action":[
      "ecs:DescribeServices",
      "ecs:CreateTaskSet",
      "ecs:DeleteTaskSet",
      "ecs:ListClusters",
      "ecs:RegisterTaskDefinition",
      "ecs:UpdateServicePrimaryTaskSet",
      "ecs:UpdateService",
      "elasticloadbalancing:DescribeTargetGroups",
      "elasticloadbalancing:DescribeListeners",
      "elasticloadbalancing:ModifyListener",
      "elasticloadbalancing:DescribeRules",
      "elasticloadbalancing:ModifyRule",
      "lambda:InvokeFunction",
      "lambda:ListFunctions",
      "cloudwatch:DescribeAlarms",
      "sns:Publish",
      "sns:ListTopics",
      "s3:GetObject",
      "s3:GetObjectVersion",
      "codedeploy:CreateApplication",
```

```
      "codedeploy:CreateDeployment",
      "codedeploy:CreateDeploymentGroup",
      "codedeploy:GetApplication",
      "codedeploy:GetDeployment",
      "codedeploy:GetDeploymentGroup",
      "codedeploy:ListApplications",
      "codedeploy:ListDeploymentGroups",
      "codedeploy:ListDeployments",
      "codedeploy:StopDeployment",
      "codedeploy:GetDeploymentTarget",
      "codedeploy:ListDeploymentTargets",
      "codedeploy:GetDeploymentConfig",
      "codedeploy:GetApplicationRevision",
      "codedeploy:RegisterApplicationRevision",
      "codedeploy:BatchGetApplicationRevisions",
      "codedeploy:BatchGetDeploymentGroups",
      "codedeploy:BatchGetDeployments",
      "codedeploy:BatchGetApplications",
      "codedeploy:ListApplicationRevisions",
      "codedeploy:ListDeploymentConfigs",
      "codedeploy:ContinueDeployment"
   ],
   "Resource":"*",
   "Effect":"Allow"
},{"Action":[
      "iam:PassRole"
   ],
   "Effect":"Allow",
   "Resource":"*",
   "Condition":{"StringLike":{"iam:PassedToService":[
            "ecs-tasks.amazonaws.com",
            "codedeploy.amazonaws.com"
         ]
      }
   }
}]
}
```

> **Note**
>
> ロールを初めて使用する場合は、リソースポリシーステートメントで次のワイルドカードを使用し、使用可能になった後にリソース名でポリシーの範囲を絞り込みます。

```
    "Resource": "*"
```

- 次のカスタム信頼ポリシー：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                 ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

このロールがアカウント接続に追加されていることを確認します。アカウント接続への IAM ロールの追加の詳細については、「」を参照してくださいアカウント接続への IAM ロールの追加。

> **Note**
>
> 必要に応じて、ここでCodeCatalystWorkflowDevelopmentRole-*spaceName*ロールの名前を指定できます。このロールの詳細については、「アカウントとスペースのCodeCatalystWorkflowDevelopmentRole-***spaceName***ロールの作成」を参照してください。CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールには、セキュリティリスクをもたらす可能性のある非常に広範なアクセス許可があることを理解します。このロールは、セキュリティが懸念されないチュートリアルやシナリオでのみ使用することをお勧めします。

対応する UI: 設定タブ/'環境/アカウント/ロール'/ロール

Inputs

(*ECSDeployAction*/Inputs)

(オプション)

`Inputs` セクションでは、ワークフローの実行中に `ECSDeployAction`が必要とするデータを定義します。

> **Note**
>
> Amazon ECS へのデプロイアクションごとに許可される入力 (ソースまたはアーティファクト) は 1 つだけです。

対応する UI: Inputs タブ

Sources

(*ECSDeployAction*/Inputs/Sources)

(タスク定義ファイルがソースリポジトリに保存されている場合に必須)

タスク定義ファイルがソースリポジトリに保存されている場合は、そのソースリポジトリのラベルを指定します。現在、サポートされているラベルは のみです`WorkflowSource`。

タスク定義ファイルがソースリポジトリに含まれていない場合は、別のアクションによって生成されたアーティファクトに存在する必要があります。

sources の詳細については、「ワークフローをソースリポジトリに接続する」を参照してください。

対応する UI: 入力タブ/ソース - オプション

Artifacts - input

(*ECSDeployAction*/Inputs/Artifacts)

(タスク定義ファイルが前のアクションの出力アーティファクトに保存されている場合に必要です)

デプロイするタスク定義ファイルが、前のアクションによって生成されたアーティファクトに含まれている場合は、ここでそのアーティファクトを指定します。タスク定義ファイルがアーティファクトに含まれていない場合は、ソースリポジトリに存在する必要があります。

アーティファクトの詳細については、「」を参照してくださいアーティファクトを使用したワークフロー内のアクション間のデータの共有。

対応する UI: 設定タブ/アーティファクト - オプション

Configuration

(*ECSDeployAction*/Configuration)

(必須)

アクションの設定プロパティを定義できるセクション。

対応する UI: 設定タブ

region

(Configuration/region)

(必須)

Amazon ECS クラスターとサービスが存在する AWS リージョンを指定します。リージョンコードのリストについては、「」の「リージョンエンドポイント」を参照してくださいAWS 全般のリファレンス。

対応する UI: 設定タブ/リージョン

cluster

(*ECSDeployAction*/Configuration/cluster)

(必須)

既存の Amazon ECS クラスターの名前を指定します。Amazon ECS にデプロイ アクションは、コンテナ化されたアプリケーションをタスクとしてこのクラスターにデプロイします。Amazon ECS クラスターの詳細については、「Amazon Elastic Container Service デベロッパーガイドhttps://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html#welcome-clusters」の「クラスター」を参照してください。

対応する UI: 設定タブ/クラスター

service

(*ECSDeployAction*/Configuration/service)

(必須)

タスク定義ファイルをインスタンス化する既存の Amazon ECS サービスの名前を指定します。このサービスは、 `cluster`フィールドで指定されたクラスターに存在する必要があります。Amazon ECS サービスの詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「Amazon ECS サービス」を参照してください。

対応する UI: 設定タブ/サービス

task-definition

(*ECSDeployAction*/Configuration/task-definition)

(必須)

既存のタスク定義ファイルへのパスを指定します。ファイルがソースリポジトリにある場合、パスはソースリポジトリのルートフォルダを基準にしています。ファイルが以前のワークフローアクションのアーティファクトに存在する場合、パスはアーティファクトルートフォルダを基準にしています。タスク定義ファイルの詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「タスク定義」を参照してください。

対応する UI: 設定タブ/タスク定義

force-new-deployment

(*ECSDeployAction*/Configuration/force-new-deployment)

(必須)

有効にすると、Amazon ECS サービスはサービス定義を変更せずに新しいデプロイを開始できます。デプロイを強制すると、サービスは現在実行中のすべてのタスクを停止し、新しいタスクを起動します。新しいデプロイの強制の詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「サービスの更新」を参照してください。

デフォルト: `false`

対応する UI: 設定タブ/サービスの新しいデプロイを強制する

codedeploy-appspec

(*ECSDeployAction*/Configuration/codedeploy-appspec)

(ブルー/グリーンデプロイを使用するように Amazon ECS サービスを設定している場合は必須、それ以外の場合は省略)

既存の CodeDeploy アプリケーション仕様 (AppSpec) ファイルの名前とパスを指定します。このファイルは、 CodeCatalyst ソースリポジトリのルートに存在する必要があります。 AppSpec ファイルの詳細については、「 ユーザーガイド」のCodeDeploy 「アプリケーション仕様 (AppSpec) ファイルAWS CodeDeploy 」を参照してください。

> **Note**
>
> ブルー/グリーンデプロイを実行するように Amazon ECS サービスを設定している場合にのみ CodeDeploy 、情報を提供してください。ローリング更新デプロイ (デフォルト) では、 CodeDeploy 情報を省略します。Amazon ECS デプロイの詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「Amazon ECS デプロイタイプ」を参照してください。

> **Note**
>
> CodeDeploy フィールドはビジュアルエディタで非表示になっている場合があります。表示するには、「」を参照してくださいビジュアルエディタに CodeDeploy フィールドがないのはなぜですか？。

対応する UI: 設定タブ/CodeDeploy AppSpec

codedeploy-application

(*ECSDeployAction*/Configuration/codedeploy-application)

(`codedeploy-appspec`が含まれている場合は必須)

既存の CodeDeploy アプリケーションの名前を指定します。 CodeDeploy アプリケーションの詳細については、「 ユーザーガイド」の「 でのアプリケーション CodeDeployの使用AWS CodeDeploy 」を参照してください。

対応する UI: 設定タブ/CodeDeploy アプリケーション

codedeploy-deployment-group

(*ECSDeployAction*/Configuration/codedeploy-deployment-group)

(codedeploy-appspecが含まれている場合は必須)

既存の CodeDeploy デプロイグループの名前を指定します。 CodeDeploy デプロイグループの詳細については、「 ユーザーガイド」の「 でのデプロイグループ CodeDeployの使用AWS CodeDeploy 」を参照してください。

対応する UI: 設定タブ/CodeDeploy デプロイグループ

codedeploy-deployment-description

(*ECSDeployAction*/Configuration/codedeploy-deployment-description)

(オプション)

このアクションが作成するデプロイの説明を指定します。詳細については、「 AWS CodeDeploy ユーザーガイド」の「 でのデプロイ CodeDeployの使用」を参照してください。

対応する UI: 設定タブ/CodeDeploy デプロイの説明

ワークフローを使用した Amazon Elastic Kubernetes Service へのアプリケーションのデプロイ

> **Tip**
>
> Kubernetes クラスターへのデプロイアクションの使用方法を示すチュートリアルについては、「」を参照してくださいチュートリアル: Amazon EKS にアプリケーションをデプロイする。

このセクションでは、 CodeCatalyst ワークフローを使用してコンテナ化されたアプリケーションを Kubernetes クラスターにデプロイする方法について説明します。これを実現するには、Kubernetes クラスターへのデプロイアクションをワークフローに追加する必要があります。このアクションは、1 つ以上の Kubernetes マニフェストファイルを使用して Amazon Elastic Kubernetes Service (EKS) で設定した Kubernetes クラスターにアプリケーションをデプロイします。サンプルマニフェストについては、deployment.yaml「」の「」を参照してくださいチュートリアル: Amazon EKS にアプリケーションをデプロイする。

Kubernetes の詳細については、「Kubernetes ドキュメント」を参照してください。

Amazon EKS の詳細については、「Amazon EKS ユーザーガイド」の「Amazon EKS とは」を参照してください。

「Deploy to Kubernetes cluster」アクションの仕組み

Kubernetes クラスターへのデプロイは次のように機能します。

1. 実行時に、 アクションは、アクションが実行されているコンピューティングマシンに CodeCatalyst Kubernetes `kubectl`ユーティリティをインストールします。アクションは、アクションの設定時に指定した Amazon EKS クラスターを指す`kubectl`ように を設定します。`kubectl apply` 次に コマンドを実行するには、 `kubectl`ユーティリティが必要です。
2. アクションは `kubectl apply -f` *`my-manifest.yaml`* コマンドを実行します*`###### my-manifest.yaml`* の手順を実行して、アプリケーションを一連のコンテナとポッドとして設定済みのクラスターにデプロイします。このコマンドの詳細については、Kubernetes リファレンスドキュメントの「kubectl apply」トピックを参照してください。

トピック

- チュートリアル: Amazon EKS にアプリケーションをデプロイする
- 「Kubernetes クラスターへのデプロイ」アクションの追加
- 「Kubernetes クラスターにデプロイ」アクションによって生成される変数
- 「Kubernetes クラスターにデプロイ」アクション YAML 定義

チュートリアル: Amazon EKS にアプリケーションをデプロイする

このチュートリアルでは、Amazon CodeCatalyst ワークフロー、Amazon EKS、およびその他のいくつかの サービスを使用して、コンテナ化されたアプリケーションを Amazon Elastic Kubernetes Service にデプロイする方法について説明します AWS 。デプロイされたアプリケーションはシンプルな「Hello, World!」です。Apache ウェブサーバーの Docker イメージ上に構築された ウェブサイト。このチュートリアルでは、開発マシンや Amazon EKS クラスターのセットアップなど、必要な準備作業を順を追って説明し、アプリケーションを構築してクラスターにデプロイするワークフローを作成する方法について説明します。

最初のデプロイが完了すると、チュートリアルではアプリケーションソースを変更するように指示します。この変更により、新しい Docker イメージが構築され、新しいリビジョン情報とともに Docker イメージリポジトリにプッシュされます。その後、Docker イメージの新しいリビジョンが Amazon EKS にデプロイされます。

> **Tip**
>
> このチュートリアルを進める代わりに、完全な Amazon EKS セットアップを実行するブループリントを使用できます。EKS アプリデプロイの設計図を使用する必要があります。詳細については、「設計図を使用したプロジェクトの作成」を参照してください。

トピック

前提条件

このチュートリアルを開始する前に：

- 接続された AWS アカウントを持つ Amazon CodeCatalyst スペースが必要です。詳細については、「スペースの作成」を参照してください。
- スペースには、次のような空の「最初から開始 CodeCatalyst」プロジェクトが必要です。

  ```
  codecatalyst-eks-project
  ```

  詳細については、「Amazon での空のプロジェクトの作成 CodeCatalyst」を参照してください。
- プロジェクトには、次の名前の空の CodeCatalyst ソースリポジトリが必要です。

  ```
  codecatalyst-eks-source-repository
  ```

詳細については、「でソースリポジトリを使用してコードを保存し、共同作業する CodeCatalyst」を参照してください。

- プロジェクトには、 CodeCatalyst 次の CI/CD 環境 (開発環境ではない) が必要です。

```
codecatalyst-eks-environment
```

この環境を次のように設定します。

- 非本番環境の など、任意のタイプを選択します。
- AWS アカウントを接続します。

詳細については、「環境を使用して AWS アカウント および VPCsにデプロイする CodeCatalyst」を参照してください。

ステップ 1: 開発マシンをセットアップする

このチュートリアルの最初のステップは、このチュートリアル全体で使用するいくつかのツールを使用して開発マシンを設定することです。これらのツールは次のとおりです。

- `eksctl` ユーティリティ – クラスター作成用
- `kubectl` ユーティリティ – の前提条件 `eksctl`
- AWS CLI - の前提条件でもあります。 `eksctl`

これらのツールがある場合は、既存の開発マシンにインストールすることも、クラウドベースの CodeCatalyst 開発環境を使用することもできます。 CodeCatalyst 開発環境の利点は、スピンアップとスピンダウンが容易で、他の CodeCatalyst サービスと統合されているため、このチュートリアルをより少ないステップで実行できます。

このチュートリアルでは、 CodeCatalyst 開発環境を使用することを前提としています。

次の手順では、 CodeCatalyst 開発環境をすばやく起動して必要なツールを使用して設定する方法を説明しますが、詳細な手順が必要な場合は、以下を参照してください。

- このガイドの「開発環境の作成」を参照してください。
- 「Amazon EKS ユーザーガイド」の「kubectl のインストール」。
- 「Amazon EKS ユーザーガイド」の「eksctl のインストールまたはアップグレード」。

- ユーザーガイドの の最新バージョンをインストールまたは更新する AWS CLIAWS Command Line Interface 。

開発環境を起動するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクト に移動しますcodecatalyst-eks-project。
3. ナビゲーションペインで [コード] を選択してから、[ソースリポジトリ] を選択します。
4. ソースリポジトリの名前 を選択しますcodecatalyst-eks-source-repository。
5. 上部近くで開発環境の作成 を選択し、AWS Cloud9 （ ブラウザで ） を選択します。
6. 既存のブランチとメインで作業が選択されていることを確認してから、「 の作成」を選択します。

   開発環境が新しいブラウザタブで起動し、リポジトリ (codecatalyst-eks-source-repository) がそこにクローンされます。

kubectl をインストールして設定するには

1. 開発環境ターミナルで、次のように入力します。

```
curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.18.9/2020-11-02/bin/linux/amd64/kubectl
```

2. 次のように入力します。

```
chmod +x ./kubectl
```

3. 次のように入力します。

```
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$PATH:$HOME/bin
```

4. 次のように入力します。

```
echo 'export PATH=$PATH:$HOME/bin' >> ~/.bashrc
```

5. 次のように入力します。

```
kubectl version --short --client
```

6. バージョンが表示されていることを確認します。

   これで、 がインストールされましたkubectl。

eksctl をインストールして設定するには

> **Note**
>
> eksctl はkubectl、代わりに を使用できるため、必須ではありません。ただし、eksctlにはクラスター設定の多くを自動化する利点があるため、このチュートリアルではが推奨されています。

1. 開発環境ターミナルで、次のように入力します。

```
curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp
```

2. 次のように入力します。

```
sudo cp /tmp/eksctl /usr/bin
```

3. 次のように入力します。

```
eksctl version
```

4. バージョンが表示されていることを確認します。

   これで、 がインストールされましたeksctl。

AWS CLI がインストールされていることを確認するには

1. 開発環境ターミナルで、次のように入力します。

```
aws --version
```

2. AWS CLI がインストールされていることを確認するために、バージョンが表示されていることを確認します。

残りの手順を完了して、 にアクセスするために必要なアクセス許可 AWS CLI を持つ を設定します AWS。

を設定するには AWS CLI

AWS サービスへのアクセスを許可するには、 アクセスキーとセッショントークン AWS CLI を使用して を設定する必要があります。次の手順では、キーとトークンを簡単に設定することができますが、詳細な手順が必要な場合は、「 ユーザーガイド」の「 の設定 AWS CLIAWS Command Line Interface 」を参照してください。

1. IAM Identity Center ユーザーを次のように作成します。

   a. にサインイン AWS Management Console し、https://console.aws.amazon.com/singlesignon/ で AWS IAM Identity Center コンソールを開きます。

      (IAM Identity Center にサインインしたことがない場合は、Enable を選択する必要があります)。

      > **Note**
      >
      > スペース AWS アカウント に接続されている CodeCatalyst を使用してサインインしていることを確認してください。接続されているアカウントを確認するには、スペースに移動し、AWS アカウントタブを選択します。詳細については、「スペースの作成」を参照してください。

   b. ナビゲーションペインで [Users] (ユーザー)、[Add user] (ユーザーの追加) の順に選択します。

   c. ユーザー名 で、次のように入力します。

      ```
      codecatalyst-eks-user
      ```

   d. パスワード で、このユーザーと共有できるワンタイムパスワードの生成 を選択します。

   e. E メールアドレス と E メールアドレスの確認 で、IAM Identity Center にまだ存在しない E メールアドレスを入力します。

   f. 名 で、次のように入力します。

      ```
      codecatalyst-eks-user
      ```

g. 姓 で、次のように入力します。

```
codecatalyst-eks-user
```

h. 表示名 で、次の状態を維持します。

```
codecatalyst-eks-user codecatalyst-eks-user
```

i. [次へ] をクリックします。

j. ユーザーをグループに追加ページで、次へ を選択します。

k. ユーザーの確認と追加ページで情報を確認し、ユーザーの追加を選択します。

ワンタイムパスワードダイアログボックスが表示されます。

l. 「コピー」を選択し、サインイン情報をテキストファイルに貼り付けます。サインイン情報は、 AWS アクセスポータル URL、ユーザー名、ワンタイムパスワードで構成されます。

m. [閉じる] を選びます。

2. アクセス許可セットを次のように作成します。

a. ナビゲーションペインで [アクセス許可セット] を選択し、[アクセス許可セットの作成] を選択します。

b. 定義済みのアクセス許可セットを選択し、 を選択しますAdministratorAccess。このポリシーは、すべての に完全なアクセス許可を提供します AWS のサービス。

c. [次へ] をクリックします。

d. アクセス許可セット名 で、 を削除AdministratorAccessして入力します。

```
codecatalyst-eks-permission-set
```

e. [次へ] をクリックします。

f. [確認と作成] ページで情報を確認し、[グループの作成] を選択します。

3. 次のようにcodecatalyst-eks-user、アクセス許可セットを に割り当てます。

a. ナビゲーションペインで を選択しAWS アカウント、現在サインイン AWS アカウント している の横にあるチェックボックスをオンにします。

b. 「ユーザーまたはグループを割り当て」を選択します。

c. [ユーザー] タブを選択します。

d. [codecatalyst-eks-user] のチェックボックスをオンにします。

e. [次へ] をクリックします。

f. [codecatalyst-eks-permission-set] のチェックボックスをオンにします。

g. [次へ] をクリックします。

h. 情報を確認し、[送信] を選択します。

これで、 codecatalyst-eks-userと codecatalyst-eks-permission-setを に割り当て AWS アカウント、それらをバインドしました。

4. 次のように、 codecatalyst-eks-userのアクセスキーとセッショントークンを取得します。

a. AWS アクセスポータル URL と、 のユーザー名とワンタイムパスワードがあることを確認しますcodecatalyst-eks-user。以前にこの情報をテキストエディタにコピーしておく必要があります。

> **Note**
>
> この情報がない場合は、IAM Identity Center codecatalyst-eks-userの詳細ページに移動し、パスワードのリセット、ワンタイムパスワードの生成 [...]、パスワードのリセットをもう一度選択して、画面に情報を表示します。

b. からサインアウトします AWS。

c. AWS アクセスポータル URL をブラウザのアドレスバーに貼り付けます。

d. 次の方法でサインインします。

- ユーザー名：

```
codecatalyst-eks-user
```

- パスワード：

one-time-password

e. 「新しいパスワードを設定する」で、新しいパスワードを入力し、「新しいパスワードを設定する」を選択します。

画面に AWS アカウント ボックスが表示されます。

f. を選択しAWS アカウント、codecatalyst-eks-userユーザーとアクセス許可セット AWS アカウント を割り当てた の名前を選択します。

g. の横にある コマンドラインまたはプログラムによるアクセス codecatalyst-eks-permission-setを選択します。

h. ページの中央にあるコマンドをコピーします。これらは次のようになります。

```
export AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
export AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
export AWS_SESSION_TOKEN="session-token"
```

セッション####は長いランダム文字列です。

5. 次のように AWS CLI、アクセスキーとセッショントークンを に追加します。

a. CodeCatalyst 開発環境に戻ります。

b. ターミナルプロンプトで、コピーしたコマンドを貼り付けます。[Enter] キーを押します。

これで、 アクセスキーとセッショントークン AWS CLI を使用して を設定しました。を使用して AWS CLI 、このチュートリアルに必要なタスクを完了できるようになりました。

> **Important**
>
> このチュートリアルのどの時点でも、次のようなメッセージが表示されます。
> `Unable to locate credentials. You can configure credentials by running "aws configure".`
> または:
> `ExpiredToken: The security token included in the request is expired`
> AWS CLI セッションの有効期限が切れているためです。この場合、 `aws configure` コマンドを実行しないでください。代わりに、 で始まるこの手順のステップ 4 の手順を使用して`Obtain codecatalyst-eks-user's access key and session token`、セッションを更新します。

ステップ 2: Amazon EKS クラスターを作成する

このセクションでは、Amazon EKS でクラスターを作成します。以下の手順では、 を使用してクラスターを作成する簡単な方法について説明しますが`eksctl`、詳細な手順が必要な場合は、以下を参照してください。

- Amazon EKS ユーザーガイドの「eksctl の開始方法」

または

- コンソールの開始方法と AWS CLI「Amazon EKS ユーザーガイド」(このトピックでは、クラスターの作成kubectl手順について説明します)

> **Note**
>
> プライベートクラスターは、Amazon EKS と CodeCatalyst の統合ではサポートされていません。

開始する前に

開発マシンで次のタスクが完了していることを確認します。

- eksctl ユーティリティをインストールしました。
- kubectl ユーティリティをインストールしました。
- をインストール AWS CLI し、アクセスキーとセッショントークンを使用して設定しました。

これらのタスクを完了する方法については、「」を参照してくださいステップ 1: 開発マシンをセットアップする。

クラスターを作成するには

> **Important**
>
> クラスターが正しく設定されないため、Amazon EKS サービスのユーザーインターフェイスを使用してクラスターを作成しないでください。次の手順で説明するように、 eksctlユーティリティを使用します。

1. 開発環境に移動します。
2. クラスターとノードを作成します。

```
eksctl create cluster --name codecatalyst-eks-cluster --region us-west-2
```

コードの説明は以下のとおりです。

- *codecatalyst-eks-cluster* は、クラスターに付ける名前に置き換えられます。
- *us-west-2* はお客様の リージョンに置き換えられます。

10～20 分後、次のようなメッセージが表示されます。

```
EKS cluster "codecatalyst-eks-cluster" in "us-west-2" region is ready
```

> **Note**
>
> がクラスターを作成すると AWS 、複数のwaiting for CloudFormation stackメッセージが表示されます。これは通常の動作です。

3. クラスターが正常に作成されたことを確認します。

```
kubectl cluster-info
```

クラスターが正常に作成されたことを示す次のようなメッセージが表示されます。

```
Kubernetes master is running at https://long-string.gr7.us-west-2.eks.amazonaws.com
CoreDNS is running at https://long-string.gr7.us-west-2.eks.amazonaws.com/api/v1/
namespaces/kube-system/services/kube-dns:dns/proxy
```

ステップ 3: Amazon ECR イメージリポジトリを作成する

このセクションでは、Amazon Elastic Container Registry (Amazon ECR) にプライベートイメージリポジトリを作成します。このリポジトリには、チュートリアル用の Docker イメージが保存されます。

Amazon ECRの詳細については、Amazon Elastic Container Registry User Guideを参照してください。

Amazon ECR でイメージリポジトリを作成するには

1. 開発環境に移動します。
2. Amazon ECR で空のリポジトリを作成します。

```
aws ecr create-repository --repository-name codecatalyst-eks-image-repo
```

を Amazon ECR リポジトリに付ける名前*codecatalyst-eks-image-repo*に置き換えます。

このチュートリアルでは、リポジトリに という名前を付けますcodecatalyst-eks-image-repo。

3. Amazon ECR リポジトリの詳細を表示します。

```
aws ecr describe-repositories \
      --repository-names codecatalyst-eks-image-repo
```

4. “repositoryUri”: 値に注意してください。例えば、 です111122223333.dkr.ecr.us-west-2.amazonaws.com/codecatalyst-eks-image-repo。

これは、後でリポジトリをワークフローに追加するときに必要になります。

ステップ 4: ソースファイルを追加する

このセクションでは、アプリケーションのソースファイルをソースリポジトリ () に追加しますcodecatalyst-eks-source-repository。これらは以下で構成されます。

- index.html ファイル — 「Hello, World!」を表示します。ブラウザの メッセージ。
- Dockerfile – Docker イメージに使用するベースイメージと、それに適用する Docker コマンドについて説明します。
- deployment.yaml ファイル — Kubernetes サービスとデプロイを定義する Kubernetes マニフェスト。

フォルダ構造は次のとおりです。

```
|– codecatalyst-eks-source-repository
   |– Kubernetes
      |– deployment.yaml
   |– public-html
   |  |– index.html
   |– Dockerfile
```

トピック

- index.html
- Dockerfile

- deployment.yaml

index.html

index.html ファイルには「Hello, World!」と表示されます。ブラウザの メッセージ。

index.html ファイルを追加するには

1. 開発環境に移動します。
2. でcodecatalyst-eks-source-repository、 という名前のフォルダを作成しますpublic-html。
3. で/public-html、次の内容index.htmlの というファイルを作成します。

```
<html>
  <head>
    <title>Hello World</title>
    <style>
      body {
      background-color: black;
      text-align: center;
      color: white;
      font-family: Arial, Helvetica, sans-serif;
      }
    </style>
  </head>
  <body>
    <h1>Hello, World!</h1>
  </body>
</html>
```

4. ターミナルプロンプトで、次のように入力します。

```
cd /projects/codecatalyst-eks-source-repository
```

5. 追加、コミット、プッシュ：

```
git add .
git commit -m "add public-html/index.html"
git push
```

index.html は、 public-htmlフォルダのリポジトリに追加されます。

Dockerfile

Dockerfile は、使用するベース Docker イメージと、それに適用する Docker コマンドを記述します。Dockerfile の詳細については、「Dockerfile リファレンス」を参照してください。

ここで指定された Dockerfile は、Apache 2.4 ベースイメージ (httpd) を使用することを示します。また、ウェブページを提供する Apache サーバーのフォルダindex.htmlに というソースファイルをコピーする手順も含まれています。Dockerfile の EXPOSE命令は、コンテナがポート 80 でリッスンしていることを Docker に伝えます。

Dockerfile を追加するには

1. でcodecatalyst-eks-source-repository、次の内容Dockerfileの というファイルを作成します。

   ```
   FROM httpd:2.4
   COPY ./public-html/index.html /usr/local/apache2/htdocs/index.html
   EXPOSE 80
   ```

   ファイル拡張子を含めないでください。

   > **Important**
   >
   > Dockerfile はリポジトリのルートフォルダに存在する必要があります。ワークフローの Docker build コマンドは、ワークフローが存在することを期待します。

2. 追加、コミット、プッシュ：

   ```
   git add .
   git commit -m "add Dockerfile"
   git push
   ```

   Dockerfile がリポジトリに追加されます。

deployment.yaml

このセクションでは、リポジトリにdeployment.yamlファイルを追加します。deployment.yaml ファイルは、実行する 2 つの Kubernetes リソースタイプまたは種類を定義する Kubernetes マニフェストです。「サービス」と「デプロイ」です。

- 「service」はロードバランサーを Amazon EC2 にデプロイします。ロードバランサーには、「Hello, World!」を参照するために使用できるインターネット向けパブリック URL と標準ポート (ポート 80) が用意されています。アプリケーションをデプロイします。
- 「deployment」は 3 つのポッドをデプロイし、各ポッドには「Hello, World!」という名前の Docker コンテナが含まれます。アプリケーションをデプロイします。3 つのポッドは、クラスターの作成時に作成されたノードにデプロイされます。

このチュートリアルのマニフェストは短いですが、マニフェストにはポッド、ジョブ、イングレス、ネットワークポリシーなど、任意の数の Kubernetes リソースタイプを含めることができます。さらに、デプロイが複雑な場合は、複数のマニフェストファイルを使用できます。

deployment.yaml ファイルを追加するには

1. でcodecatalyst-eks-source-repository、 という名前のフォルダを作成しますKubernetes。
2. で/Kubernetes、次の内容deployment.yamlの というファイルを作成します。

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
  labels:
    app: my-app
spec:
  type: LoadBalancer
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
  labels:
    app: my-app
spec:
  replicas: 3
```

```
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: codecatalyst-eks-container
        # The $REPOSITORY_URI and $IMAGE_TAG placeholders will be replaced by
 actual values supplied by the build action in your workflow
        image: $REPOSITORY_URI:$IMAGE_TAG
        ports:
        - containerPort: 80
```

3. 追加、コミット、プッシュ：

```
git add .
git commit -m "add Kubernetes/deployment.yaml"
git push
```

deployment.yaml ファイルは、 というフォルダのリポジトリに追加されますKubernetes。

これで、すべてのソースファイルが追加されました。

作業内容を再度確認し、すべてのファイルを正しいフォルダに配置したことを確認してください。フォルダ構造は次のとおりです。

```
|— codecatalyst-eks-source-repository
   |— Kubernetes
      |— deployment.yaml
   |— public-html
   |  |— index.html
   |— Dockerfile
```

ステップ 5: AWS ロールを作成する

このセクションでは、 CodeCatalyst ワークフローが機能するために必要な AWS IAM ロールを作成します。これらのロールは次のとおりです。

- ビルドロール – AWS アカウントにアクセスして Amazon ECR と Amazon EC2 に書き込むための CodeCatalyst ビルドアクション (ワークフロー内) 許可を付与します。
- ロールのデプロイ — AWS アカウントと Amazon EKS CodeCatalyst にアクセスするための (ワークフロー内の) Kubernetes クラスターへのデプロイアクションのアクセス許可を付与します。

IAM ロールの詳細については、「 ユーザーガイド」の「IAM ロールAWS Identity and Access Management 」を参照してください。

> **Note**
>
> 時間を節約するために、前述の 2 つのロールではなく、CodeCatalystWorkflowDevelopmentRole-*spaceName*ロールと呼ばれる 1 つのロールを作成できます。詳細については、「アカウントとスペースのCodeCatalystWorkflowDevelopmentRole-***spaceName***ロールの作成」を参照してください。CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールには、セキュリティリスクをもたらす可能性のある非常に広範なアクセス許可があることを理解します。このロールは、セキュリティが懸念されないチュートリアルやシナリオでのみ使用することをお勧めします。このチュートリアルでは、前述の 2 つのロールを作成することを前提としています。

ビルドロールとデプロイロールを作成するには、以下の一連の手順を実行します。

1. 両方のロールの信頼ポリシーを作成するには

1. 開発環境に移動します。
2. Cloud9-*long-string* ディレクトリで、次の内容codecatalyst-eks-trust-policy.jsonで という名前のファイルを作成します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
```

```
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

2. ビルドロールのビルドポリシーを作成するには

- Cloud9-*long-string* ディレクトリで、次の内容codecatalyst-eks-build-policy.jsonで という名前のファイルを作成します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:*",
                "ec2:*"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> ロールを初めて使用してワークフローアクションを実行するときは、リソースポリシーステートメントでワイルドカードを使用し、使用可能になった後にリソース名でポリシーの範囲を絞り込みます。
>
> ```
> "Resource": "*"
> ```

3. デプロイロールのデプロイポリシーを作成するには

- Cloud9-*long-string* ディレクトリで、次の内容codecatalyst-eks-deploy-policy.jsonで という名前のファイルを作成します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeCluster",
                "eks:ListClusters"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> ロールを初めて使用してワークフローアクションを実行するときは、リソースポリシーステートメントでワイルドカードを使用し、使用可能になった後にリソース名でポリシーの範囲を絞り込みます。
>
> ```
> "Resource": "*"
> ```

これで、開発環境に 3 つのポリシードキュメントが追加されました。ディレクトリ構造は次のようになります。

```
|— Cloud9-long-string
   |— .c9
   |— codecatalyst-eks-source-repository
      |— Kubernetes
      |— public-html
      |— Dockerfile
   codecatalyst-eks-build-policy.json
   codecatalyst-eks-deploy-policy.json
   codecatalyst-eks-trust-policy.json
```

4. ビルドポリシーを に追加するには AWS

1. 開発環境ターミナルで、次のように入力します。

```
cd /projects
```

2. 次のように入力します。

```
aws iam create-policy \
    --policy-name codecatalyst-eks-build-policy \
    --policy-document file://codecatalyst-eks-build-policy.json
```

3. [Enter] キーを押します。
4. コマンド出力で、 などの"arn":値を書き留めますarn:aws:iam::111122223333:policy/codecatalyst-eks-build-policy。この ARN は後で必要になります。

5. デプロイポリシーを に追加するには AWS

1. 次のように入力します。

```
aws iam create-policy \
    --policy-name codecatalyst-eks-deploy-policy \
    --policy-document file://codecatalyst-eks-deploy-policy.json
```

2. [Enter] キーを押します。
3. コマンド出力で、デプロイポリシーの値"arn":、例えば を書き留めますarn:aws:iam::111122223333:policy/codecatalyst-eks-deploy-policy。この ARN は後で必要になります。

6. ビルドロールを作成するには

1. 次のように入力します。

```
aws iam create-role \
      --role-name codecatalyst-eks-build-role \
      --assume-role-policy-document file://codecatalyst-eks-trust-policy.json
```

2. [Enter] キーを押します。
3. 次のように入力します。

```
aws iam attach-role-policy \
      --role-name codecatalyst-eks-build-role \
```

```
      --policy-arn arn:aws:iam::111122223333:policy/codecatalyst-eks-build-policy
```

ここで*#arn:aws:iam::111122223333:policy/codecatalyst-eks-build-policy*は、前にメモしたビルドポリシーの ARN に置き換えられます。

4. [Enter] キーを押します。
5. ターミナルプロンプトで、次のように入力します。

```
aws iam get-role \
      --role-name codecatalyst-eks-build-role
```

6. [Enter] キーを押します。
7. ロール"Arn":の値、例えば を書き留めますarn:aws:iam::111122223333:role/codecatalyst-eks-build-role。この ARN は後で必要になります。

7. デプロイロールを作成するには

1. 次のように入力します。

```
aws iam create-role \
      --role-name codecatalyst-eks-deploy-role \
      --assume-role-policy-document file://codecatalyst-eks-trust-policy.json
```

2. [Enter] キーを押します。
3. 次のように入力します。

```
aws iam attach-role-policy \
      --role-name codecatalyst-eks-deploy-role \
      --policy-arn arn:aws:iam::111122223333:policy/codecatalyst-eks-deploy-policy
```

ここで*#arn:aws:iam::111122223333:policy/codecatalyst-eks-deploy-policy*は、前にメモしたデプロイポリシーの ARN に置き換えられます。

4. [Enter] キーを押します。
5. 次のように入力します。

```
aws iam get-role \
      --role-name codecatalyst-eks-deploy-role
```

6. [Enter] キーを押します。

7. ロールの値"Arn":、例えば を書き留めますarn:aws:iam::111122223333:role/codecatalyst-eks-deploy-role。この ARN は後で必要になります。

これで、ビルドロールとデプロイロールを作成し、その ARNs。

ステップ 6: に AWS ロールを追加する CodeCatalyst

このステップでは、スペースに接続 AWS アカウント した にビルドロール (codecatalyst-eks-build-role) とデプロイロール (codecatalyst-eks-deploy-role) を追加します。これにより、ロールをワークフローで使用できるようになります。

ビルドロールとデプロイロールを に追加するには AWS アカウント

1. CodeCatalyst コンソールで、スペースに移動します。
2. 上部で、設定 を選択します。
3. ナビゲーションペインで、AWS アカウント を選択します。アカウントのリストが表示されます。
4. Amazon CodeCatalyst の表示名列で、ビルドロールとデプロイロールを作成した AWS アカウント の表示名をコピーします。(数字である可能性があります)。この値は、後でワークフローを作成するときに必要になります。
5. 表示名を選択します。
6. AWS 管理コンソール からロールの管理 を選択します。

   「Amazon CodeCatalyst スペースへの IAM ロールの追加」ページが表示されます。ページにアクセスするには、サインインが必要な場合があります。
7. 「IAM で作成した既存のロールを追加」を選択します。

   ドロップダウンリストが表示されます。このリストには、ビルドロールとデプロイロール、および codecatalyst-runner.amazonaws.comとcodecatalyst.amazonaws.comサービスプリンシパルを含む信頼ポリシーを持つ他の IAM ロールが表示されます。
8. ドロップダウンリストから、以下を追加します。

   - codecatalyst-eks-build-role
   - codecatalyst-eks-deploy-role

> **Note**
>
> が表示された場合はThe security token included in the request is invalid、適切なアクセス許可がないためである可能性があります。この問題を解決するには、 からサインアウト AWS して、スペースの作成時に使用した AWS アカウントで再度サインインします CodeCatalyst。

9. CodeCatalyst コンソールに戻り、ページを更新します。

   これで、ビルドロールとデプロイロールが IAM ロール の下に表示されます。

   これらのロールが CodeCatalyst ワークフローで使用できるようになりました。

ステップ 7: を更新する ConfigMap

で作成したデプロイロールステップ 5: AWS ロールを作成するを Kubernetes ConfigMap ファイルに追加して、Kubernetes クラスターへのデプロイアクション (ワークフロー内) にクラスターにアクセスして操作できるようにする必要があります。eksctl または を使用してkubectl、このタスクを実行できます。

eksctl を使用して Kubernetes ConfigMap ファイルを設定するには

- 開発環境ターミナルで、次のように入力します。

```
eksctl create iamidentitymapping --cluster codecatalyst-eks-cluster --
arn arn:aws:iam::111122223333:role/codecatalyst-eks-deploy-role --group
 system:masters --username codecatalyst-eks-deploy-role --region us-west-2
```

  コードの説明は以下のとおりです。

  - *codecatalyst-eks-cluster* は、Amazon EKS クラスターのクラスター名に置き換えられます。
  - *arn:aws:iam::111122223333:role/codecatalyst-eks-deploy-role* は、 で作成したデプロイロールの ARN に置き換えられますステップ 5: AWS ロールを作成する。
  - *codecatalyst-eks-deploy-role* (の隣--username) は、 で作成したデプロイロールの名前に置き換えられますステップ 5: AWS ロールを作成する。

> **Note**
>
> デプロイロールを作成しない場合は、をCodeCatalystWorkflowDevelopmentRole-*spaceName*ロールの名前*codecatalyst-eks-deploy-role*に置き換えます。このロールの詳細については、「ステップ 5: AWS ロールを作成する」を参照してください。

- *us-west-2* はお客様の リージョンに置き換えられます。

このコマンドの詳細については、「IAM ユーザーとロールの管理」を参照してください。

次のようなメッセージが表示されます。

```
2023-06-09 00:58:29 [#]  checking arn arn:aws:iam::111122223333:role/codecatalyst-
eks-deploy-role against entries in the auth ConfigMap
2023-06-09 00:58:29 [#]  adding identity "arn:aws:iam::111122223333:role/
codecatalyst-eks-deploy-role" to auth ConfigMap
```

kubectl を使用して Kubernetes ConfigMap ファイルを設定するには

1. 開発環境ターミナルで、次のように入力します。

```
kubectl edit configmap -n kube-system aws-auth
```

ConfigMap ファイルが画面に表示されます。

2. テキストを赤色の斜体で追加します。

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file
 will be
# reopened with the relevant failures.
#
apiVersion: v1
data:
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
```

```
      rolearn: arn:aws:iam::111122223333:role/eksctl-codecatalyst-eks-cluster-n-
NodeInstanceRole-16BC456ME6YR5
      username: system:node:{{EC2PrivateDNSName}}
    - groups:
      - system:masters
      rolearn: arn:aws:iam::111122223333:role/codecatalyst-eks-deploy-role
      username: codecatalyst-eks-deploy-role
  mapUsers: |
    []
kind: ConfigMap
metadata:
  creationTimestamp: "2023-06-08T19:04:39Z"
  managedFields:
  ...
```

コードの説明は以下のとおりです。

- *arn:aws:iam::111122223333:role/codecatalyst-eks-deploy-role* は、 で作成したデプロイロールの ARN に置き換えられますステップ 5: AWS ロールを作成する。
- *codecatalyst-eks-deploy-role* (の横username:) は、 で作成したデプロイロールの名前に置き換えられますステップ 5: AWS ロールを作成する。

> **Note**
>
> デプロイロールを作成しない場合は、をCodeCatalystWorkflowDevelopmentRole-*spaceName*ロールの名前*codecatalyst-eks-deploy-role*に置き換えます。このロールの詳細については、「ステップ 5: AWS ロールを作成する」を参照してください。

詳細については、「Amazon EKS ユーザーガイド」の「クラスターへの IAM プリンシパルアクセスの有効化」を参照してください。

これで、デプロイロール、さらには Amazon EKS へのデプロイアクション、Kubernetes クラスターへのアクセスsystem:masters許可が付与されました。

ステップ 8: ワークフローを作成して実行する

このステップでは、ソースファイルを取得し、Docker イメージにビルドして、Amazon EKS クラスターのツリーポッドにイメージをデプロイするワークフローを作成します。

ワークフローは、順番に実行される次の構成要素で構成されます。

- トリガー — このトリガーは、変更をソースリポジトリにプッシュすると、ワークフローの実行を自動的に開始します。トリガーについての詳細は、「トリガーを使用したワークフローの自動実行の開始」を参照してください。
- ビルドアクション (BuildBackend) – トリガー時に、 アクションは Dockerfile を使用して Docker イメージをビルドし、そのイメージを Amazon ECR にプッシュします。また、ビルドアクションは、 deployment.yaml ファイル内の $REPOSITORY_URI変数と $IMAGE_TAG変数を正しい値で更新し、このファイルと Kubernetesフォルダ内のその他の出力アーティファクトを作成します。このチュートリアルでは、フォルダ内の唯一のファイルは Kubernetes ですがdeployment.yaml、さらにファイルを含めることができます。アーティファクトはデプロイアクションの入力として使用され、次は です。

  ビルドアクションの詳細については、「」を参照してくださいワークフローによる構築。
- デプロイアクション (DeployToEKS) – ビルドアクションが完了すると、デプロイアクションはビルドアクション (Manifests) によって生成された出力アーティファクトを検索し、その中に deployment.yaml ファイルを見つけます。次に、 アクションは deployment.yaml ファイルの指示に従って 3 つのポッドを実行します。それぞれに 1 つの「Hello, World!」が含まれます。Docker コンテナ — Amazon EKS クラスター内。

ワークフローを作成するには

1. CodeCatalyst コンソールに移動します。
2. プロジェクト () に移動しますcodecatalyst-eks-project。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの作成 を選択します。
5. ソースリポジトリ で、 を選択しますcodecatalyst-eks-source-repository。
6. ブランチ で、 を選択しますmain。
7. [作成] を選択します。
8. YAML サンプルコードを削除します。
9. 次の YAML コードを追加して、新しいワークフロー定義ファイルを作成します。

> **Note**
>
> ワークフロー定義ファイルの詳細については、「」を参照してくださいワークフロー YAML 定義。

```
Name: codecatalyst-eks-workflow
SchemaVersion: 1.0

Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  BuildBackend:
    Identifier: aws/build@v1
    Environment:
      Name: codecatalyst-eks-environment
      Connections:
        - Name: codecatalyst-account-connection
          Role: codecatalyst-eks-build-role
    Inputs:
      Sources:
        - WorkflowSource
      Variables:
        - Name: REPOSITORY_URI
          Value: 111122223333.dkr.ecr.us-west-2.amazonaws.com/codecatalyst-eks-
image-repo
        - Name: IMAGE_TAG
          Value: ${WorkflowSource.CommitId}
    Configuration:
      Steps:
        #pre_build:
        - Run: echo Logging in to Amazon ECR...
        - Run: aws --version
        - Run: aws ecr get-login-password --region us-west-2 | docker login --
username AWS --password-stdin 111122223333.dkr.ecr.us-west-2.amazonaws.com
        #build:
        - Run: echo Build started on `date`
        - Run: echo Building the Docker image...
        - Run: docker build -t $REPOSITORY_URI:latest .
```

```
        - Run: docker tag $REPOSITORY_URI:latest $REPOSITORY_URI:$IMAGE_TAG
        #post_build:
        - Run: echo Build completed on `date`
        - Run: echo Pushing the Docker images...
        - Run: docker push $REPOSITORY_URI:latest
        - Run: docker push $REPOSITORY_URI:$IMAGE_TAG
        # Replace the variables in deployment.yaml
        - Run: find Kubernetes/ -type f | xargs sed -i "s|\$REPOSITORY_URI|
$REPOSITORY_URI|g"
        - Run: find Kubernetes/ -type f | xargs sed -i "s|\$IMAGE_TAG|$IMAGE_TAG|g"
        - Run: cat Kubernetes/*
        # The output artifact will be a zip file that contains Kubernetes manifest
 files.
    Outputs:
      Artifacts:
        - Name: Manifests
          Files:
            - "Kubernetes/*"
  DeployToEKS:
    DependsOn:
      - BuildBackend
    Identifier: aws/kubernetes-deploy@v1
    Environment:
      Name: codecatalyst-eks-environment
      Connections:
        - Name: codecatalyst-account-connection
          Role: codecatalyst-eks-deploy-role
    Inputs:
      Artifacts:
        - Manifests
    Configuration:
      Namespace: default
      Region: us-west-2
      Cluster: codecatalyst-eks-cluster
      Manifests: Kubernetes/
```

上記のコードで、以下を置き換えます。

- で作成した環境の名前*codecatalyst-eks-environment*を持つ の両方のインスタンス前提条件。

- アカウント接続の表示名*codecatalyst-account-connection*を持つ の両方のインスタンス。表示名は数字にすることができます。詳細については、「ステップ 6: に AWS ロールを追加する CodeCatalyst」を参照してください。
- *codecatalyst-eks-build-role* で作成したビルドロールの名前。 ステップ 5: AWS ロールを作成する
- *111122223333.dkr.ecr.us-west-2.amazonaws.com/codecatalyst-eks-image-repo* (`Value:`プロパティ内) と、 で作成した Amazon ECR リポジトリの URIステップ 3: Amazon ECR イメージリポジトリを作成する。
- イメージサフィックス () のない Amazon ECR リポジトリの URI を持つ *111122223333.dkr.ecr.us-west-2.amazonaws.com* (`Run: aws ecr` コマンド内) / `codecatalyst-eks-image-repo`。
- *codecatalyst-eks-deploy-role* に、 で作成したデプロイロールの名前を入力しますステップ 5: AWS ロールを作成する。
- AWS リージョンコードを含む *us-west-2* の両方のインスタンス。リージョンコードのリストについては、「」の「リージョンエンドポイント」を参照してくださいAWS 全般のリファレンス。

> **Note**
>
> ビルドロールとデプロイロールを作成しない場合は、 *codecatalyst-eks-build-role* と をCodeCatalystWorkflowDevelopmentRole-*spaceName*ロールの名前*codecatalyst-eks-deploy-role*に置き換えます。このロールの詳細については、「ステップ 5: AWS ロールを作成する」を参照してください。

10. (オプション) 検証 を選択して、コミットする前に YAML コードが有効であることを確認します。
11. [Commit] (コミット) を選択します。
12. コミットワークフローダイアログボックスに、次のように入力します。

    a. コミットメッセージ で、テキストを削除して次のように入力します。

    ```
    Add first workflow
    ```

    b. リポジトリ で、 を選択しますcodecatalyst-eks-source-repository。

    c. ブランチ名 で、main を選択します。

d. [Commit] (コミット) を選択します。

これでワークフローが作成されました。ワークフロー実行は、ワークフローの上部で定義されたトリガーが原因で自動的に開始されます。具体的には、workflow.yamlファイルをソースリポジトリにコミット (およびプッシュ) すると、トリガーによってワークフロー実行が開始されます。

ワークフロー実行の進行状況を表示するには

1. CodeCatalyst コンソールのナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
2. 先ほど作成したワークフロー を選択しますcodecatalyst-eks-workflow。
3. ビルドの進行状況BuildBackendを表示するには、 を選択します。
4. DeployToEKS を選択すると、デプロイの進行状況が表示されます。

実行の詳細の表示の詳細については、「」を参照してくださいワークフローの実行ステータスと詳細の表示。

デプロイを確認するには

1. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
2. 左側の下部近くで、ロードバランサー を選択します。
3. Kubernetes デプロイの一部として作成されたロードバランサーを選択します。どのロードバランサーを選択するかわからない場合は、タグタブで次のタグを探します。
   - kubernetes.io/service-name
   - kubernetes.io/cluster/ekstutorialcluster
4. 正しいロードバランサーを選択し、説明タブを選択します。
5. DNS 名の値をコピーしてブラウザのアドレスバーに貼り付けます。

「Hello, World!」 ウェブページがブラウザに表示され、アプリケーションが正常にデプロイされたことを示します。

ステップ 9: ソースファイルを変更する

このセクションでは、ソースリポジトリの index.html ファイルを変更します。この変更により、ワークフローは新しい Docker イメージを構築し、コミット ID でタグ付けして Amazon ECR にプッシュし、Amazon ECS にデプロイします。

index.html を変更するには

1. 開発環境に移動します。
2. ターミナルプロンプトで、 をソースリポジトリに変更します。

```
cd /projects/codecatalyst-eks-source-repository
```

3. 最新のワークフロー変更を取得します。

```
git pull
```

4. codecatalyst-eks-source-repository/public-html/index.html を開きます。
5. 14 行目で、Hello, World!テキストを に変更しますTutorial complete!。
6. 追加、コミット、プッシュ :

```
git add .
git commit -m "update index.html title"
git push
```

   ワークフロー実行が自動的に開始されます。

7. (オプション) 次のように入力します。

```
git show HEAD
```

   index.html 変更のコミット ID を書き留めます。このコミット ID は、先ほど開始したワークフロー実行によってデプロイされる Docker イメージにタグ付けされます。

8. デプロイの進行状況を確認します。

   a. CodeCatalyst コンソールのナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。

   b. を選択して、最新の実行codecatalyst-eks-workflowを表示します。

c. BuildBackend、、および DeployToEKS を選択して、ワークフローの実行の進行状況を確認します。

9. 次のように、アプリケーションが更新されていることを確認します。

a. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。

b. 左側の下部近くで、ロードバランサー を選択します。

c. Kubernetes デプロイの一部として作成されたロードバランサーを選択します。

d. DNS 名の値をコピーしてブラウザのアドレスバーに貼り付けます。

「チュートリアルが完了しました！」 ウェブページがブラウザに表示され、アプリケーションの新しいリビジョンが正常にデプロイされたことを示します。

10. (オプション) で AWS Amazon ECR コンソールに切り替え、新しい Docker イメージがこの手順のステップ 7 のコミット ID でタグ付けされていることを確認します。

クリーンアップ

このチュートリアルで使用するストレージとコンピューティングリソースに対して不必要に課金されないように、環境をクリーンアップする必要があります。

次をクリーンアップするには：

1. クラスターを削除します。

- 開発環境ターミナルで、次のように入力します。

```
eksctl delete cluster --region=us-west-2 --name=codecatalyst-eks-cluster
```

コードの説明は以下のとおりです。

- *us-west-2* はお客様の リージョンに置き換えられます。
- *codecatalyst-eks-cluster* は、作成したクラスターの名前に置き換えられます。

5 ~ 10 分後、クラスターおよび関連するリソースは削除されます。これには、 AWS CloudFormation スタック、ノードグループ (Amazon EC2)、ロードバランサーが含まれますが、これらに限定されません。

> **Important**
>
> `eksctl delete cluster` コマンドが機能しない場合は、 AWS 認証情報または `kubectl`認証情報の更新が必要になる場合があります。更新する認証情報が不明な場合は、まず AWS 認証情報を更新します。 AWS 認証情報を更新するには、「」を参照してください「認証情報が見つかりません」と「」のExpiredTokenエラーを修正するにはどうすればよいですか？。`kubectl` 認証情報を更新するには、「」を参照してください「サーバーに接続できません」というエラーを修正するにはどうすればよいですか？。

2. AWS コンソールで、次のようにクリーンアップします。

   1. Amazon ECR で、 を削除します`codecatalyst-eks-image-repo`。
   2. IAM Identity Center で、以下を削除します。
      a. `codecatalyst-eks-user`
      b. `codecatalyst-eks-permission-set`
   3. IAM で、以下を削除します。
      - `codecatalyst-eks-build-role`
      - `codecatalyst-eks-deploy-role`
      - `codecatalyst-eks-build-policy`
      - `codecatalyst-eks-deploy-policy`

3. CodeCatalyst コンソールで、次のようにクリーンアップします。

   1. を削除します`codecatalyst-eks-workflow`。
   2. を削除します`codecatalyst-eks-environment`。
   3. を削除します`codecatalyst-eks-source-repository`。
   4. 開発環境を削除します。
   5. を削除します`codecatalyst-eks-project`。

このチュートリアルでは、 CodeCatalyst ワークフローと Kubernetes クラスターへのデプロイ アクションを使用して Amazon EKS サービスにアプリケーションをデプロイする方法を学習しました。

「Kubernetes クラスターへのデプロイ」アクションの追加

次の手順を使用して、Kubernetes クラスターへのデプロイアクションをワークフローに追加します。

開始する前に

ワークフローに Kubernetes クラスターへのデプロイ アクションを追加する前に、次の準備が必要です。

> **Tip**
>
> これらの前提条件をすばやく設定するには、「」の手順に従いますチュートリアル: Amazon EKS にアプリケーションをデプロイする。

- Amazon EKS の Kubernetes クラスター。クラスターの詳細については、「Amazon EKS ユーザーガイド」の「Amazon EKS クラスター」を参照してください。
- アプリケーションを Docker イメージにアセンブルする方法を説明する Dockerfile が少なくとも 1 つあります。Dockerfiles の詳細については、「Dockerfile リファレンス」を参照してください。
- 少なくとも 1 つの Kubernetes マニフェストファイル。Kubernetes ドキュメントの設定ファイルまたは設定と呼ばれます。詳細については、Kubernetes ドキュメントの「リソースの管理」を参照してください。
- Kubernetes クラスターへのデプロイアクションに Amazon EKS クラスターにアクセスして操作する機能を付与する IAM ロール。詳細については、「Kubernetes クラスターにデプロイ」アクション YAML 定義 の Role トピックを参照してください。

  このロールを作成したら、以下に追加する必要があります。

  - Kubernetes ConfigMap ファイル。ファイルにロールを追加する方法については、 ConfigMap「Amazon EKS ユーザーガイド」の「クラスターへの IAM プリンシパルアクセスの有効化」を参照してください。
  - CodeCatalyst。IAM ロールを に追加する方法については、 CodeCatalyst「」を参照してくださいアカウント接続への IAM ロールの追加。
- CodeCatalyst スペース、プロジェクト、環境。スペースと環境はどちらも、アプリケーションをデプロイする AWS アカウントに接続する必要があります。詳細については、スペースの作成、Amazon での空のプロジェクトの作成 CodeCatalyst、および環境を使用して AWS アカウント および VPCsにデプロイする CodeCatalyst を参照してください。

- でサポートされているソースリポジトリ CodeCatalyst。リポジトリには、アプリケーションのソースファイル、Dockerfile、Kubernetes マニフェストが保存されます。詳細については、「でソースリポジトリを使用してコードを保存し、共同作業する CodeCatalyst」を参照してください。

Visual

ビジュアルエディタを使用して「Kubernetes クラスターへのデプロイ」アクションを追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. ビジュアル を選択します。
7. 左上で + Actions を選択してアクションカタログを開きます。
8. ドロップダウンリストから Amazon CodeCatalystを選択します。
9. Kubernetes クラスターへのデプロイアクションを検索し、次のいずれかを実行します。
   - プラス記号 (+) を選択してワークフロー図にアクションを追加し、設定ペインを開きます。

     または

   - Kubernetes クラスターにデプロイ を選択します。アクションの詳細ダイアログボックスが表示されます。このダイアログボックスで、次の操作を行います。
     - (オプション) ダウンロード を選択して、アクションのソースコード を表示します。
     - ワークフローに追加 を選択して、ワークフロー図にアクションを追加し、その設定ペインを開きます。
10. 入力タブと設定タブで、必要に応じてフィールドに入力します。各フィールドの説明については、「」を参照してください「Kubernetes クラスターにデプロイ」アクション YAML 定義。このリファレンスでは、YAML エディタとビジュアルエディタの両方に表示される各フィールド (および対応する YAML プロパティ値) に関する詳細情報を提供します。

11. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
12. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

YAML

YAML エディタを使用して「Kubernetes クラスターへのデプロイ」アクションを追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. YAML を選択します。
7. 左上で + Actions を選択してアクションカタログを開きます。
8. ドロップダウンリストから Amazon CodeCatalystを選択します。
9. Kubernetes クラスターへのデプロイアクションを検索し、次のいずれかを実行します。
   - プラス記号 (+) を選択してワークフロー図にアクションを追加し、設定ペインを開きます。

     または

   - Kubernetes クラスターにデプロイ を選択します。アクションの詳細ダイアログボックスが表示されます。このダイアログボックスで、次の操作を行います。
     - (オプション) ダウンロード を選択して、アクションのソースコード を表示します。
     - ワークフローに追加 を選択して、ワークフロー図にアクションを追加し、その設定ペインを開きます。
10. 必要に応じて YAML コードのプロパティを変更します。使用可能な各プロパティの説明は、「」に記載されています「Kubernetes クラスターにデプロイ」アクション YAML 定義。
11. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
12. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

「Kubernetes クラスターにデプロイ」アクションによって生成される変数

Kubernetes クラスターにデプロイ アクションは、実行時に次の変数を生成して設定します。これらは事前定義された変数 と呼ばれます。

ワークフローでこれらの変数を参照する方法については、「」を参照してください事前定義された変数の使用。

キー	値
クラスター	ワークフローの実行中にデプロイされた Amazon EKS クラスターの の Amazon.com リソースネーム (ARN)。 例: `arn:aws:eks:us-west-2:111122223333:cluster/codecatalyst-eks-cluster`
デプロイプラットフォーム	デプロイプラットフォームの名前。 にハードコードされます`AWS:EKS`。
metadata	リザーブド。ワークフローの実行中にデプロイされたクラスターに関連する JSON 形式のメタデータ。
名前空間	クラスターがデプロイされた Kubernetes 名前空間。 例: `default`
resources	リザーブド。ワークフローの実行中にデプロイされたリソースに関連する JSON 形式のメタデータ。
server	などの管理ツールを使用してクラスターとの通信に使用できる API サーバーエンドポイントの名前`kubectl`。

キー	値
	API サービスエンドポイントの詳細については、「Amazon EKS ユーザーガイド」の「Amazon EKS クラスターエンドポイントのアクセスコントロール」を参照してください。 例: `https://random-string .gr7.us-west-2.eks.amazonaws.com`

「Kubernetes クラスターにデプロイ」アクション YAML 定義

以下は、Kubernetes クラスターへのデプロイアクションの YAML 定義です。このアクションの使用方法については、「」を参照してくださいワークフローを使用した Amazon Elastic Kubernetes Service へのアプリケーションのデプロイ。

このアクション定義は、より広範なワークフロー定義ファイル内のセクションとして存在します。ファイルの詳細については、「ワークフロー YAML 定義」を参照してください。

> **Note**
>
> 後続の YAML プロパティのほとんどには、対応する UI 要素がビジュアルエディタにあります。UI 要素を検索するには、Ctrl+F を使用します。 要素は、関連付けられた YAML プロパティとともに一覧表示されます。

```
# The workflow definition starts here.
# See ######### for details.

Name: MyWorkflow
SchemaVersion: 1.0
Actions:

# The action definition starts here.
  DeployToKubernetesCluster_nn:
    Identifier: aws/kubernetes-deploy@v1
    DependsOn:
      - build-action
    Compute:
```

```
        - Type: EC2 | Lambda
        - Fleet: fleet-name
    Timeout: timeout-minutes
    Environment:
      Name: environment-name
      Connections:
        - Name: account-connection-name
          Role: DeployToEKS
    Inputs:
      # Specify a source or an artifact, but not both.
      Sources:
        - source-name-1
      Artifacts:
        - manifest-artifact
    Configuration:
      Namespace: namespace
      Region: us-east-1
      Cluster: eks-cluster
      Manifests: manifest-path
```

DeployToKubernetesCluster

(必須)

アクションの名前を指定します。すべてのアクション名は、ワークフロー内で一意である必要があります。アクション名は、英数字 (a～z、A～Z、0～9)、ハイフン (-)、アンダースコア (_) に制限されています。スペースは使用できません。引用符を使用してアクション名で特殊文字やスペースを有効にすることはできません。

デフォルト: DeployToKubernetesCluster_nn。

対応する UI: 設定タブ/アクションの表示名

Identifier

(*DeployToKubernetesCluster*/Identifier)

(必須)

アクションを識別します。バージョンを変更しない限り、このプロパティを変更しないでください。詳細については、「アクションのメジャー、マイナー、またはパッチバージョンの指定」を参照してください。

デフォルト: aws/kubernetes-deploy@v1。

対応する UI: ワークフロー図/DeployToKubernetesCluster_nn/aws/kubernetes-deploy@v1 ラベル

DependsOn

(*DeployToKubernetesCluster*/DependsOn)

(オプション)

このアクションを実行するために正常に実行する必要があるアクション、アクショングループ、またはゲートを指定します。

「依存」機能の詳細については、「」を参照してください他のアクションに依存するようにアクションを設定する。

対応する UI: の入力タブ/依存 - オプション

Compute

(*DeployToKubernetesCluster*/Compute)

(オプション)

ワークフローアクションを実行するために使用されるコンピューティングエンジン。コンピューティングはワークフローレベルまたはアクションレベルで指定できますが、両方を指定することはできません。ワークフローレベルで指定すると、コンピューティング設定はワークフローで定義されたすべてのアクションに適用されます。ワークフローレベルでは、同じインスタンスで複数のアクションを実行することもできます。詳細については、「アクション間でのコンピューティングの共有」を参照してください。

対応する UI: なし

Type

(*DeployToKubernetesCluster*/Compute/Type)

(Computeが含まれている場合は必須)

コンピューティングエンジンのタイプ。次のいずれかの値を使用できます。

- EC2 (ビジュアルエディタ) または EC2 (YAML エディタ)

アクション実行中の柔軟性のために最適化されました。

- Lambda (ビジュアルエディタ) または Lambda (YAML エディタ)

  アクションの起動速度を最適化しました。

コンピューティングタイプの詳細については、「コンピューティングタイプ」を参照してください。

対応する UI: 設定タブ/詳細 - オプション/コンピューティングタイプ

Fleet

(*DeployToKubernetesCluster*/Compute/Fleet)

(オプション)

ワークフローまたはワークフローアクションを実行するマシンまたはフリートを指定します。オンデマンドフリートでは、アクションが開始されると、ワークフローは必要なリソースをプロビジョニングし、アクションが終了するとマシンは破棄されます。オンデマンドフリートの例: `Linux.x86-64.Large`、`Linux.x86-64.XLarge`。オンデマンドフリートの詳細については、「」を参照してくださいオンデマンドフリートのプロパティ。

プロビジョニングされたフリートでは、ワークフローアクションを実行するように一連の専用マシンを設定します。これらのマシンはアイドル状態のままで、すぐにアクションを処理できます。プロビジョニングされたフリートの詳細については、「」を参照してくださいプロビジョニングされたフリートのプロパティ。

`Fleet` を省略した場合、デフォルトは ですLinux.x86-64.Large。

対応する UI: 設定タブ/詳細 - オプション/コンピューティングフリート

Timeout

(*DeployToKubernetesCluster*/Timeout)

(オプション)

CodeCatalyst アクションが終了するまでに実行できる時間を分単位で指定します (YAML エディタ)、または時間と分単位で指定します (ビジュアルエディタ)。最小値は 5 分で、最大値は「」で説明されていますワークフローのクォータ。デフォルトのタイムアウトは、最大タイムアウトと同じです。

対応する UI: 設定タブ/タイムアウト - オプション

Environment

(*DeployToKubernetesCluster*/Environment)

(必須)

アクションで使用する CodeCatalyst 環境を指定します。

環境の詳細については、環境を使用して AWS アカウント および VPCsにデプロイする CodeCatalyst 「」および「」を参照してください環境を作成する。

対応する UI: 設定タブ/'環境/アカウント/ロール'/環境

Name

(*DeployToKubernetesCluster*/Environment/Name)

(Environmentが含まれている場合は必須)

アクションに関連付ける既存の環境の名前を指定します。

対応する UI: 設定タブ/'環境/アカウント/ロール'/環境

Connections

(*DeployToKubernetesCluster*/Environment/Connections)

(Environmentが含まれている場合は必須)

アクションに関連付けるアカウント接続を指定します。で最大 1 つのアカウント接続を指定できますEnvironment。

アカウント接続の詳細については、「」を参照してください接続された AWS リソースへのアクセスを許可する AWS アカウント。アカウント接続を環境に関連付ける方法については、「」を参照してください環境を作成する。

対応する UI: 設定タブ/'環境/アカウント/ロール'/AWS アカウント接続

Name

(*DeployToKubernetesCluster*/Environment/Connections/Name)

(必須)

アカウント接続の名前を指定します。

対応する UI: 設定タブ/'環境/アカウント/ロール'/AWS アカウント接続

Role

(*DeployToKubernetesCluster*/Environment/Connections/Role)

(必須)

Kubernetes クラスターへのデプロイアクションが へのアクセスに使用する IAM ロールの名前を指定します AWS。このロールに次のポリシーが含まれていることを確認します。

- 次のアクセス許可ポリシー：

> **Warning**
>
> アクセス許可を次のポリシーに示すものに制限します。より広範なアクセス許可を持つロールを使用すると、セキュリティ上のリスクが生じる可能性があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeCluster",
                "eks:ListClusters"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> ロールを初めて使用する場合は、リソースポリシーステートメントで次のワイルドカードを使用し、使用可能になった後にリソース名でポリシーの範囲を絞り込みます。

```
    "Resource": "*"
```

- 次のカスタム信頼ポリシー：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                 ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

このロールが以下に追加されていることを確認します。

- アカウント接続。アカウント接続への IAM ロールの追加の詳細については、「」を参照してくださいアカウント接続への IAM ロールの追加。
- Kubernetes ConfigMap。IAM ロールを に追加する方法の詳細については ConfigMap、 `eksctl`ドキュメントの「IAM ユーザーとロールの管理」を参照してください。

> **Tip**
>
> アカウント接続に IAM ロールを追加する手順については、「」およびチュートリアル: Amazon EKS にアプリケーションをデプロイする「」も参照してください ConfigMap。

> **Note**
>
> 必要に応じて、ここでCodeCatalystWorkflowDevelopmentRole-*spaceName*ロールの名前を指定できます。このロールの詳細については、「アカウントとスペースのCodeCatalystWorkflowDevelopmentRole-***spaceName***ロールの作成」を参照してください。CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールには、セキュリティリスクをもたらす可能性のある非常に広範なアクセス許可があることを理解します。このロールは、セキュリティが懸念されないチュートリアルとシナリオでのみ使用することをお勧めします。CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールを使用する場合は、 eksctlドキュメントの「IAM ユーザーとロールの管理」の手順に従って、ConfigMap 必ずファイルに追加してください。

対応する UI: 設定タブ/'環境/アカウント/ロール'/ロール

Inputs

(*DeployToKubernetesCluster*/Inputs)

(オプション)

Inputs セクションでは、ワークフローの実行中に がDeployToKubernetesCluster必要とするデータを定義します。

> **Note**
>
> Amazon EKS へのデプロイアクションごとに許可される入力 (ソースまたはアーティファクト) は 1 つだけです。

対応する UI: Inputs タブ

Sources

(*DeployToKubernetesCluster*/Inputs/Sources)

(マニフェストファイルがソースリポジトリに保存されている場合に必須)

Kubernetes マニフェストファイルまたはファイルがソースリポジトリに保存されている場合は、そのソースリポジトリのラベルを指定します。現在、サポートされているラベルは のみですWorkflowSource。

マニフェストファイルがソースリポジトリに含まれていない場合は、別のアクションによって生成されたアーティファクトに存在する必要があります。

sources の詳細については、「ワークフローをソースリポジトリに接続する」を参照してください。

対応する UI: 入力タブ/ソース - オプション

Artifacts - input

(*DeployToKubernetesCluster*/Inputs/Artifacts)

(マニフェストファイルが前のアクションの出力アーティファクトに保存されている場合に必要です)

Kubernetes マニフェストファイルまたはファイルが前のアクションによって生成されたアーティファクトに含まれている場合は、ここでそのアーティファクトを指定します。マニフェストファイルがアーティファクトに含まれていない場合は、ソースリポジトリに存在する必要があります。

アーティファクトの例などの詳細については、「」を参照してくださいアーティファクトを使用したワークフロー内のアクション間のデータの共有。

対応する UI: 設定タブ/アーティファクト - オプション

Configuration

(*DeployToKubernetesCluster*/Configuration)

(必須)

アクションの設定プロパティを定義できるセクション。

対応する UI: 設定タブ

Namespace

(*DeployToKubernetesCluster*/Configuration/Namespace)

(オプション)

Kubernetes アプリケーションをデプロイする Kubernetes 名前空間を指定します。クラスターで名前空間を使用していない場合`default`は、 を使用します。名前空間の詳細については、Kubernetes ドキュメントの「Kubernetes 名前空間を使用したクラスターの分割」を参照してください。

名前空間を省略すると、 の値`default`が使用されます。

対応する UI: 設定タブ/名前空間

Region

(*DeployToKubernetesCluster*/Configuration/Region)

(必須)

Amazon EKS クラスターとサービスが存在する AWS リージョンを指定します。リージョンコードのリストについては、「」の「リージョンエンドポイント」を参照してくださいAWS 全般のリファレンス。

対応する UI: 設定タブ/リージョン

Cluster

(*DeployToKubernetesCluster*/Configuration/Cluster)

(必須)

既存の Amazon EKS クラスターの名前を指定します。Kubernetes クラスターにデプロイ アクションは、コンテナ化されたアプリケーションをこのクラスターにデプロイします。Amazon EKS クラスターの詳細については、「Amazon EKS ユーザーガイド」の「クラスター」を参照してください。

対応する UI: 設定タブ/クラスター

Manifests

(*DeployToKubernetesCluster*/Configuration/Manifests)

(必須)

YAML 形式の Kubernetes マニフェストファイルへのパスを指定します (設定ファイル 、設定ファイル 、または単に設定と呼ばれます)。

複数のマニフェストファイルを使用している場合は、それらを 1 つのフォルダに配置し、そのフォルダを参照します。マニフェストファイルは Kubernetes によって英数字で処理されるため、ファイル名の前に必ず数字や文字を増やして処理順序を制御してください。例:

```
00-namespace.yaml
```

```
01-deployment.yaml
```

マニフェストファイルがソースリポジトリにある場合、パスはソースリポジトリのルートフォルダを基準にしています。ファイルが以前のワークフローアクションのアーティファクトに存在する場合、パスはアーティファクトルートフォルダを基準にしています。

例:

```
Manifests/

deployment.yaml

my-deployment.yml
```

ワイルドカード () は使用しないでください*。

> **Note**
>
> Helm チャートと kustomization ファイルはサポートされていません。

マニフェストファイルの詳細については、Kubernetes ドキュメントの「リソース設定の整理」を参照してください。

対応する UI: 設定タブ/マニフェスト

ワークフローを使用した AWS CloudFormation スタックのデプロイ

このセクションでは、 CodeCatalyst ワークフローを使用して AWS CloudFormation スタックをデプロイする方法について説明します。これを実現するには、 AWS CloudFormation スタックのデプロイアクションをワークフローに追加する必要があります。アクションは、指定したテンプレート AWS に基づいて、リソースの CloudFormation スタックを にデプロイします。テンプレートは、次のようになります。

- AWS CloudFormation template – 詳細については、「テンプレートの使用 AWS CloudFormation」を参照してください。
- AWS SAM テンプレート – 詳細については、AWS Serverless Application Model 「 (AWS SAM) 仕様」を参照してください。

> **Note**
>
> AWS SAM テンプレートを使用するには、まず `sam package`オペレーションを使用して AWS SAM アプリケーションをパッケージ化する必要があります。Amazon CodeCatalyst

ワークフローの一部としてこのパッケージを自動的に実行する方法を示すチュートリアルについては、「」を参照してくださいチュートリアル: を使用してサーバーレスアプリケーションをデプロイする AWS CloudFormation。

スタックがすでに存在する場合、 アクションは CloudFormation `CreateChangeSet`オペレーションを実行し、次に `ExecuteChangeSet`オペレーションを実行します。その後、 アクションは変更がデプロイされるのを待ち、結果に応じて、失敗した場合は成功としてマークします。

デプロイするリソースを含む AWS CloudFormation または AWS SAM テンプレートが既にある場合、または AWS SAM や などのツールを使用してワークフロービルドアクションの一部として自動的に生成する予定がある場合は、 AWS CloudFormation スタックのデプロイアクションを使用しますAWS Cloud Development Kit (AWS CDK)。 ???

テンプレートには、作成できるものや AWS CloudFormation スタックのデプロイアクション CloudFormation AWS SAM で使用できるものに制限はありません。

> **Tip**
>
> AWS CloudFormation スタックのデプロイアクションを使用してサーバーレスアプリケーションをデプロイする方法を示すチュートリアルについては、「」を参照してくださいチュートリアル: を使用してサーバーレスアプリケーションをデプロイする AWS CloudFormation。

トピック

- チュートリアル: を使用してサーバーレスアプリケーションをデプロイする AWS CloudFormation
- AWS CloudFormation 「スタックのデプロイ」アクションの追加
- ロールバックの設定
- AWS CloudFormation 「スタックのデプロイ」アクションによって生成される変数
- AWS CloudFormation 「スタックのデプロイ」アクション YAML 定義

チュートリアル: を使用してサーバーレスアプリケーションをデプロイする AWS CloudFormation

このチュートリアルでは、ワークフローを使用してサーバーレスアプリケーションを CloudFormation スタックとして構築、テスト、デプロイする方法について説明します。

このチュートリアルのアプリケーションは、「Hello World」メッセージを出力するシンプルなウェブアプリケーションです。これは AWS Lambda 関数と Amazon API Gateway で構成され、の拡張機能である AWS Serverless Application Model (AWS SAM) を使用して構築しますAWS CloudFormation。

トピック

前提条件

開始する前に:

- 接続された AWS アカウントを持つ CodeCatalyst スペースが必要です。詳細については、「スペースの作成」を参照してください。
- スペースには、次のような空の「最初から開始 CodeCatalyst」プロジェクトが必要です。

  ```
  codecatalyst-cfn-project
  ```

  詳細については、「Amazon での空のプロジェクトの作成 CodeCatalyst」を参照してください。
- プロジェクトには、次の CodeCatalyst 環境が必要です。

  ```
  codecatalyst-cfn-environment
  ```

  この環境を次のように設定します。
  - 非本番環境の など、任意のタイプを選択します。
  - アカウントをその AWS アカウントに接続します。

詳細については、「環境を使用して AWS アカウント および VPCsにデプロイする CodeCatalyst」を参照してください。

ステップ 1: ソースリポジトリを作成する

このステップでは、 でソースリポジトリを作成します CodeCatalyst。このリポジトリは、Lambda 関数ファイルなどのチュートリアルのソースファイルを保存するために使用されます。

ソースリポジトリの詳細については、「」を参照してくださいソースリポジトリの作成。

ソースリポジトリを作成するには

1. で CodeCatalyst、ナビゲーションペインでコード を選択し、ソースリポジトリ を選択します。
2. [リポジトリの追加] を選択し、[リポジトリの作成] を選択します。
3. リポジトリ名 で、次のように入力します。

```
codecatalyst-cfn-source-repository
```

4. [作成] を選択します。

これで、 というリポジトリが作成されましたcodecatalyst-cfn-source-repository。

ステップ 2: AWS ロールを作成する

このステップでは、次の AWS IAM ロールを作成します。

- ロールのデプロイ — サーバーレスアプリケーションをデプロイする AWS アカウントと CloudFormation サービスにアクセスするためのデプロイ CodeCatalyst AWS CloudFormation スタックアクションのアクセス許可を付与します。 AWS CloudFormation スタックのデプロイアクションはワークフローの一部です。
- ビルドロール – AWS アカウントにアクセスし、サーバーレスアプリケーションパッケージが保存される Amazon S3 に書き込むための CodeCatalyst ビルドアクションのアクセス許可を付与します。ビルドアクションはワークフローの一部です。
- スタックロール – 後で指定する AWS SAM テンプレートで指定されたリソースを読み取って変更する CloudFormation アクセス許可を付与します。また、 にアクセス許可を付与します CloudWatch。

IAM ロールの詳細については、「 ユーザーガイド」の「IAM ロールAWS Identity and Access Management 」を参照してください。

> **Note**
>
> 時間を節約するために、前述の 3 つのロールではなく、CodeCatalystWorkflowDevelopmentRole-*spaceName*ロールと呼ばれる 1 つのロールを作成できます。詳細については、「アカウントとスペースのCodeCatalystWorkflowDevelopmentRole-***spaceName***ロールの作成」を参照してください。CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールには、セキュリティリスクをもたらす可能性のある非常に広範なアクセス許可があることを理解します。このロールは、セキュリティが懸念されないチュートリアルやシナリオでのみ使用することをお勧めします。このチュートリアルでは、前述の 3 つのロールを作成することを前提としています。

> **Note**
>
> Lambda 実行ロールも必要ですが、ステップ 5 でワークフローを実行するときにsam-template.ymlファイルが作成するため、今すぐ作成する必要はありません。

デプロイロールを作成するには

1. 次のように、ロールのポリシーを作成します。
   a. にサインインします AWS。
   b. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
   c. ナビゲーションペインで、ポリシー を選択します。
   d. ポリシーの作成を選択します。
   e. [JSON] タブを選択します。
   f. 既存のコードを削除します。
   g. 次のコードを貼り付けます。

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [{
    "Action": [
        "cloudformation:CreateStack",
        "cloudformation:DeleteStack",
        "cloudformation:Describe*",
        "cloudformation:UpdateStack",
        "cloudformation:CreateChangeSet",
        "cloudformation:DeleteChangeSet",
        "cloudformation:ExecuteChangeSet",
        "cloudformation:SetStackPolicy",
        "cloudformation:ValidateTemplate",
        "cloudformation:List*",
        "iam:PassRole"
    ],
    "Resource": "*",
    "Effect": "Allow"
}]
}
```

> **Note**
>
> ロールを初めて使用してワークフローアクションを実行するときは、リソースポリシーステートメントでワイルドカードを使用し、使用可能になった後にリソース名でポリシーの範囲を絞り込みます。
>
> ```
> "Resource": "*"
> ```

h. [次へ : タグ] を選択します。

i. [次へ: レビュー] を選択します。

j. 名前 で、次のように入力します。

```
codecatalyst-deploy-policy
```

k. [ポリシーの作成] を選択します。

これで、アクセス許可ポリシーが作成されました。

2. デプロイロールを次のように作成します。

a. ナビゲーションペインで ロール を選択してから、ロールを作成する を選択します。

b. カスタム信頼ポリシー を選択します。

c. 既存のカスタム信頼ポリシーを削除します。

d. 次のカスタム信頼ポリシーを追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                 ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

e. [次へ] をクリックします。

f. アクセス許可ポリシー で、そのチェックボックスを検索codecatalyst-deploy-policyして選択します。

g. [次へ] をクリックします。

h. ロール名 には、次のように入力します。

```
codecatalyst-deploy-role
```

i. ロールの説明 には、次のように入力します。

```
CodeCatalyst deploy role
```

j. [ロールを作成] を選択します。

これで、信頼ポリシーとアクセス許可ポリシーを使用してデプロイロールが作成されました。

3. 次のように、デプロイロール ARN を取得します。

a. ナビゲーションペインで、[ロール] を選択します。

b. 検索ボックスに、作成したロールの名前 () を入力しますcodecatalyst-deploy-role。

c. リストからロールを選択します。

ロールの概要ページが表示されます。

d. 上部で、ARN 値をコピーします。

これで、適切なアクセス許可を持つデプロイロールを作成し、その ARN を取得しました。

ビルドロールを作成するには

1. 次のように、ロールのポリシーを作成します。

a. にサインインします AWS。

b. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。

c. ナビゲーションペインで、ポリシー を選択します。

d. ポリシーの作成を選択します。

e. [JSON] タブを選択します。

f. 既存のコードを削除します。

g. 次のコードを貼り付けます。

```
{
    "Version": "2012-10-17",
    "Statement": [{
    "Action": [
        "s3:PutObject",
        "iam:PassRole"
    ],
    "Resource": "*",
    "Effect": "Allow"
}]
}
```

> **Note**
>
> ロールを初めて使用してワークフローアクションを実行するときは、リソースポリシーステートメントでワイルドカードを使用し、使用可能になった後にリソース名でポリシーの範囲を絞り込みます。

```
        "Resource": "*"
```

h. [次へ : タグ] を選択します。

i. [次へ: レビュー] を選択します。

j. 名前 で、次のように入力します。

```
codecatalyst-build-policy
```

k. [ポリシーの作成] を選択します。

これで、アクセス許可ポリシーが作成されました。

2. 次のようにビルドロールを作成します。

a. ナビゲーションペインで ロール を選択してから、ロールを作成する を選択します。

b. カスタム信頼ポリシー を選択します。

c. 既存のカスタム信頼ポリシーを削除します。

d. 次のカスタム信頼ポリシーを追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

e. [次へ] をクリックします。

f. アクセス許可ポリシー で、そのチェックボックスを検索codecatalyst-build-policyして選択します。

g. [次へ] をクリックします。

h. ロール名 には、次のように入力します。

```
codecatalyst-build-role
```

i. ロールの説明 には、次のように入力します。

```
CodeCatalyst build role
```

j. [ロールを作成] を選択します。

これで、信頼ポリシーとアクセス許可ポリシーを使用してビルドロールが作成されました。

3. 次のように、ビルドロール ARN を取得します。

a. ナビゲーションペインで、[ロール] を選択します。

b. 検索ボックスに、作成したロールの名前 () を入力しますcodecatalyst-build-role。

c. リストからロールを選択します。

ロールの概要ページが表示されます。

d. 上部で、ARN 値をコピーします。

これで、適切なアクセス許可を持つビルドロールを作成し、その ARN を取得しました。

スタックロールを作成するには

1. スタックをデプロイするアカウント AWS を使用して にサインインします。
2. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
3. 次のようにスタックロールを作成します。

a. ナビゲーションペインで Roles (ロール) を選択します。

b. [ロールの作成] を選択します。

c. [AWS サービス] を選択してください。

d. ユースケースセクションCloudFormationで、ドロップダウンリストから を選択します。

e. ラジオボタン [CloudFormation] を選択します。

f. 下部で、次へ を選択します。

g. 検索ボックスを使用して、次のアクセス許可ポリシーを検索し、それぞれのチェックボックスをオンにします。

> **Note**
>
> ポリシーを検索してもポリシーが表示されない場合は、フィルターをクリアを選択して、もう一度試してください。

- CloudWatchFullAccess
- AWS CloudFormationFullAccess
- IAMFullAccess
- AWS Lambda_FullAccess
- AmazonAPIGatewayAdministrator
- AmazonS3FullAccess
- AmazonEC2ContainerRegistryFullAccess

最初のポリシーでは、アラームが発生したときにスタックのロールバックを有効にする CloudWatch ために へのアクセスを許可します。

残りのポリシーでは AWS SAM 、このチュートリアルでデプロイされるスタック内のサービスとリソースへのアクセスを に許可します。詳細については、「 AWS Serverless Application Model デベロッパーガイド」の「アクセス許可」を参照してください。

h. [次へ] をクリックします。

i. ロール名 には、次のように入力します。

```
codecatalyst-stack-role
```

j. [ロールを作成] を選択します。

4. 次のように、スタックロールの ARN を取得します。

a. ナビゲーションペインで、[ロール] を選択します。

b. 検索ボックスに、作成したロールの名前 () を入力しますcodecatalyst-stack-role。

c. リストからロールを選択します。

d. 概要セクションで、ARN 値をコピーします。これは、後で必要になります。

これで、適切なアクセス許可を持つスタックロールが作成され、その ARN が取得されました。

ステップ 3: に AWS ロールを追加する CodeCatalyst

このステップでは、ビルドロール (codecatalyst-build-role) とデプロイロール (codecatalyst-deploy-role) をスペース内の CodeCatalyst アカウント接続に追加します。

> **Note**
>
> 接続にスタックロール (codecatalyst-stack-role) を追加する必要はありません。これは、 CodeCatalyst と の間にデプロイロール AWS を使用して接続が確立された後、スタックロールが CloudFormation (ではなく CodeCatalyst) によって使用されるためです。スタックロールは、 が にアクセス CodeCatalyst するために使用しないため AWS、アカウント接続に関連付ける必要はありません。

ビルドロールとデプロイロールをアカウント接続に追加するには

1. で CodeCatalyst、スペースに移動します。
2. [AWS アカウント] を選択します。アカウント接続のリストが表示されます。
3. ビルドロールとデプロイロールを作成したアカウントを表す AWS アカウント接続を選択します。
4. AWS 管理コンソール からロールの管理 を選択します。

   「Amazon CodeCatalyst スペースへの IAM ロールの追加」ページが表示されます。ページにアクセスするには、サインインが必要な場合があります。
5. 「IAM で作成した既存のロールを追加」を選択します。

   ドロップダウンリストが表示されます。このリストには、 codecatalyst-runner.amazonaws.comおよび codecatalyst.amazonaws.comサービスプリンシパルを含む信頼ポリシーを持つすべての IAM ロールが表示されます。
6. ドロップダウンリストで、 を選択しcodecatalyst-build-role、ロールの追加 を選択します。
7. 「IAM ロールを追加」を選択し、「IAM で作成した既存のロールを追加」を選択し、ドロップダウンリストから「」を選択しますcodecatalyst-deploy-role。[Add role] を選択します。

これで、ビルドロールとデプロイロールをスペースに追加しました。

8. Amazon CodeCatalyst 表示名 の値をコピーします。この値は、後でワークフローを作成するときに必要になります。

ステップ 4: Amazon S3 バケットを作成する

このステップでは、サーバーレスアプリケーションのデプロイパッケージ .zip ファイルを保存する Amazon S3 バケットを作成します。

Amazon S3 バケットを作成するには

1. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。
2. メインペインで、バケットの作成 を選択します。
3. バケット名 には、次のように入力します。

```
codecatalyst-cfn-s3-bucket
```

4. [AWS リージョン] で、リージョンを選択します。このチュートリアルでは、米国西部 (オレゴン) us-west-2 を選択したことを前提としています。Amazon S3 でサポートされているリージョンの詳細については、「」の「Amazon Simple Storage Service エンドポイントとクォータ」を参照してくださいAWS 全般のリファレンス。
5. ページの下部で、バケットの作成 を選択します。

これで、米国西部 (オレゴン) us-west-2 リージョン**codecatalyst-cfn-s3-bucket**に という名前のバケットが作成されました。

ステップ 5: ソースファイルを追加する

このステップでは、複数のアプリケーションソースファイルを CodeCatalyst ソースリポジトリに追加します。`hello-world` フォルダには、デプロイするアプリケーションファイルが含まれています。`tests` フォルダにはユニットテストが含まれています。フォルダ構造は次のとおりです。

```
.
|— hello-world
|   |— tests
|      |— unit
|         |— test-handler.js
|   |— app.js
```

```
|— .npmignore
|— package.json
|— sam-template.yml
|— setup-sam.sh
```

.npmignore ファイル

.npmignore ファイルは、npm がアプリケーションパッケージから除外するファイルとフォルダを示します。このチュートリアルでは、npm はアプリケーションの一部ではないため、testsフォルダを除外します。

.npmignore ファイルを追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。 codecatalyst-cfn-project
3. ナビゲーションペインで [コード] を選択してから、[ソースリポジトリ] を選択します。
4. ソースリポジトリのリストから、リポジトリ を選択しますcodecatalyst-cfn-source-repository。
5. ファイル で、ファイルの作成 を選択します。
6. ファイル名 には、次のように入力します。

   ```
   .npmignore
   ```

7. テキストボックスに、次のコードを入力します。

   ```
   tests/*
   ```

8. コミット を選択し、もう一度コミット を選択します。

   これで、リポジトリのルート.npmignoreに という名前のファイルが作成されました。

package.json ファイル

package.json ファイルには、プロジェクト名、バージョン番号、説明、依存関係、およびアプリケーションとやり取りして実行する方法を説明するその他の詳細など、Node プロジェクトに関する重要なメタデータが含まれています。

このチュートリアルpackage.jsonの には、依存関係のリストとtestスクリプトが含まれています。テストスクリプトは以下を実行します。

- テストスクリプトは、mocha を使用して で指定されたユニットテストを実行しhello-world/tests/unit/、xunit レポーターを使用して結果をjunit.xmlファイルに書き込みます。
- Istanbul (nyc) を使用すると、テストスクリプトはクローバーレポーターを使用してコードカバレッジレポート (clover.xml) を生成します。詳細については、イスタンブールのドキュメントの「代替レポーターの使用」を参照してください。

package.json ファイルを追加するには

1. リポジトリのファイル で、ファイルの作成 を選択します。
2. ファイル名 には、次のように入力します。

   package.json

3. テキストボックスに、次のコードを入力します。

```
{
  "name": "hello_world",
  "version": "1.0.0",
  "description": "hello world sample for NodeJS",
  "main": "app.js",
  "repository": "https://github.com/awslabs/aws-sam-cli/tree/develop/samcli/local/
init/templates/cookiecutter-aws-sam-hello-nodejs",
  "author": "SAM CLI",
  "license": "MIT",
  "dependencies": {
    "axios": "^0.21.1",
    "nyc": "^15.1.0"
  },
  "scripts": {
    "test": "nyc --reporter=clover mocha hello-world/tests/unit/ --reporter xunit
 --reporter-option output=junit.xml"
  },
  "devDependencies": {
    "aws-sdk": "^2.815.0",
    "chai": "^4.2.0",
    "mocha": "^8.2.1"
  }
}
```

4. コミット を選択し、もう一度コミット を選択します。

これで、リポジトリのルートpackage.jsonに というファイルが追加されました。

sam-template.yml ファイル

sam-template.yml ファイルには、Lambda 関数と API Gateway をデプロイし、一緒に設定する手順が含まれています。テンプレートAWS Serverless Application Model 仕様 に従っており、 AWS CloudFormation テンプレート仕様を拡張します。

は便利な AWS::Serverless::Function リソースタイプ AWS SAM を提供するため、このチュートリアル AWS SAM では通常の AWS CloudFormation テンプレートの代わりに テンプレートを使用します。このタイプは、基本 CloudFormation 構文を使用するために通常書き出す必要がある多くの behind-the-scenes 設定を実行します。例えば、 は Lambda 関数、Lambda 実行ロール、および関数を開始するイベントソースマッピングAWS::Serverless::Functionを作成します。基本的な を使用して書き込む場合は、これらすべてをコーディングする必要があります CloudFormation。

このチュートリアルでは、事前に作成されたテンプレートを使用しますが、ビルドアクションを使用してワークフローの一部としてテンプレートを生成できます。詳細については、「ワークフローを使用した AWS CloudFormation スタックのデプロイ」を参照してください。

sam-template.yml ファイルを追加するには

1. リポジトリのファイル で、ファイルの作成 を選択します。
2. ファイル名 には、次のように入力します。

```
sam-template.yml
```

3. テキストボックスに、次のコードを入力します。

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: >
  serverless-api

  Sample SAM Template for serverless-api

# More info about Globals: https://github.com/awslabs/serverless-application-model/
blob/master/docs/globals.rst
Globals:
  Function:
    Timeout: 3
```

```
Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function # For details on this resource type,
 see https://github.com/awslabs/serverless-application-model/blob/master/
versions/2016-10-31.md#awsserverlessfunction
    Properties:
      CodeUri: hello-world/
      Handler: app.lambdaHandler
      Runtime: nodejs12.x
      Events:
        HelloWorld:
          Type: Api # For details on this event source type, see
 https://github.com/awslabs/serverless-application-model/blob/master/
versions/2016-10-31.md#api
          Properties:
            Path: /hello
            Method: get

Outputs:
  # ServerlessRestApi is an implicit API created out of the events key under
 Serverless::Function
  # Find out about other implicit resources you can reference within AWS SAM at
  # https://github.com/awslabs/serverless-application-model/blob/master/docs/
internals/generated_resources.rst#api
  HelloWorldApi:
    Description: "API Gateway endpoint URL for the Hello World function"
    Value: !Sub "https://${ServerlessRestApi}.execute-api.
${AWS::Region}.amazonaws.com/Prod/hello/"
  HelloWorldFunction:
    Description: "Hello World Lambda function ARN"
    Value: !GetAtt HelloWorldFunction.Arn
  HelloWorldFunctionIamRole:
    Description: "Implicit Lambda execution role created for the Hello World
 function"
    Value: !GetAtt HelloWorldFunctionRole.Arn
```

4. コミット を選択し、もう一度コミット を選択します。

   これで、リポジトリのルートフォルダsam-template.ymlに というファイルが追加されました。

setup-sam.sh ファイル

setup-sam.sh ファイルには、 AWS SAM CLI ユーティリティをダウンロードしてインストールする手順が含まれています。ワークフローでは、このユーティリティを使用してhello-worldソースをパッケージ化します。

setup-sam.sh ファイルを追加するには

1. リポジトリのファイル で、ファイルの作成 を選択します。
2. ファイル名 には、次のように入力します。

```
setup-sam.sh
```

3. テキストボックスに、次のコードを入力します。

```
#!/usr/bin/env bash
echo "Setting up sam"

yum install unzip -y

curl -L0 https://github.com/aws/aws-sam-cli/releases/latest/download/aws-sam-cli-
linux-x86_64.zip
unzip -qq aws-sam-cli-linux-x86_64.zip -d sam-installation-directory

./sam-installation-directory/install; export AWS_DEFAULT_REGION=us-west-2
```

   上記のコードで、*us-west-2* を自分のリージョンに置き換えます AWS 。
4. コミット を選択し、もう一度コミット を選択します。

   これで、リポジトリのルートsetup-sam.shに というファイルが追加されました。

app.js ファイル

には Lambda 関数コードapp.jsが含まれています。このチュートリアルでは、コードはテキストを返しますhello world。

app.js ファイルを追加するには

1. リポジトリのファイル で、ファイルの作成 を選択します。
2. ファイル名 には、次のように入力します。

hello-world/app.js

3. テキストボックスに、次のコードを入力します。

```
// const axios = require('axios')
// const url = 'http://checkip.amazonaws.com/';
let response;

/**
 *
 * Event doc: https://docs.aws.amazon.com/apigateway/latest/developerguide/set-up-
lambda-proxy-integrations.html#api-gateway-simple-proxy-for-lambda-input-format
 * @param {Object} event - API Gateway Lambda Proxy Input Format
 *
 * Context doc: https://docs.aws.amazon.com/lambda/latest/dg/nodejs-prog-model-
context.html
 * @param {Object} context
 *
 * Return doc: https://docs.aws.amazon.com/apigateway/latest/developerguide/set-up-
lambda-proxy-integrations.html
 * @returns {Object} object - API Gateway Lambda Proxy Output Format
 *
 */
exports.lambdaHandler = async (event, context) => {
    try {
        // const ret = await axios(url);
        response = {
            'statusCode': 200,
            'body': JSON.stringify({
                message: 'hello world',
                // location: ret.data.trim()
            })
        }
    } catch (err) {
        console.log(err);
        return err;
    }

    return response
};
```

4. コミット を選択し、もう一度コミット を選択します。

これで、 という名前のフォルダhello-worldと という名前のファイルが作成されましたapp.js。

test-handler.js ファイル

test-handler.js ファイルには、Lambda 関数の単位テストが含まれています。

test-handler.js ファイルを追加するには

1. リポジトリのファイル で、ファイルの作成 を選択します。
2. ファイル名 には、次のように入力します。

```
hello-world/tests/unit/test-handler.js
```

3. テキストボックスに、次のコードを入力します。

```
'use strict';

const app = require('../../app.js');
const chai = require('chai');
const expect = chai.expect;
var event, context;

describe('Tests index', function () {
    it('verifies successful response', async () => {
        const result = await app.lambdaHandler(event, context)

        expect(result).to.be.an('object');
        expect(result.statusCode).to.equal(200);
        expect(result.body).to.be.an('string');

        let response = JSON.parse(result.body);

        expect(response).to.be.an('object');
        expect(response.message).to.be.equal("hello world");
        // expect(response.location).to.be.an("string");
    });
});
```

4. コミット を選択し、もう一度コミット を選択します。

これで、 hello-world/tests/unitフォルダtest-handler.jsの下に というファイルが追加されました。

これで、すべてのソースファイルが追加されました。

作業内容を再度確認し、すべてのファイルを正しいフォルダに配置したことを確認してください。フォルダ構造は次のとおりです。

```
.
|— hello-world
|   |— tests
|      |— unit
|         |— test-handler.js
|   |— app.js
|— .npmignore
|— README.md
|— package.json
|— sam-template.yml
|— setup-sam.sh
```

ステップ 6: ワークフローを作成して実行する

このステップでは、Lambda ソースコードをパッケージ化してデプロイするワークフローを作成します。ワークフローは、順番に実行される次の構成要素で構成されます。

- トリガー — このトリガーは、変更をソースリポジトリにプッシュすると、ワークフローの実行を自動的に開始します。トリガーについての詳細は、「トリガーを使用したワークフローの自動実行の開始」を参照してください。
- テストアクション (Test) — トリガー時に、このアクションは Node パッケージマネージャー (npm) をインストールし、 npm run test コマンドを実行します。このコマンドは、package.json ファイルで定義されたtestスクリプトを実行するように npm に指示します。次に、testスクリプトはユニットテストを実行し、テストレポート (junit.xml) とコードカバレッジレポート () の 2 つのレポートを生成しますclover.xml。詳細については、「package.json ファイル」を参照してください。

  次に、テストアクションは XML レポートを CodeCatalyst レポートに変換し、 CodeCatalyst コンソールのテストアクションのレポートタブに表示します。

  テストアクションの詳細については、「」を参照してくださいワークフローを使用したテスト。

- ビルドアクション (BuildBackend) – テストアクションが完了すると、ビルドアクションは AWS SAM CLI をダウンロードしてインストールし、hello-worldソースをパッケージ化し、パッケージを Lambda サービスが想定する Amazon S3 バケットにコピーします。アクションは、 という新しい AWS SAM テンプレートファイルも出力sam-template-packaged.ymlし、 という出力アーティファクトに配置しますbuildArtifact。

  ビルドアクションの詳細については、「」を参照してくださいワークフローによる構築。
- デプロイアクション (DeployCloudFormationStack) – ビルドアクションが完了すると、デプロイアクションはビルドアクション (buildArtifact) によって生成された出力アーティファクトを検索し、その中に AWS SAM テンプレートを見つけて、テンプレートを実行します。 AWS SAM テンプレートは、サーバーレスアプリケーションをデプロイするスタックを作成します。

ワークフローを作成するには

1. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
2. ワークフローの作成 を選択します。
3. ソースリポジトリ で、 を選択しますcodecatalyst-cfn-source-repository。
4. ブランチ で、 を選択しますmain。
5. [作成] を選択します。
6. YAML サンプルコードを削除します。
7. 次の YAML コードを追加します。

```
Name: codecatalyst-cfn-workflow
SchemaVersion: 1.0

Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  Test:
    Identifier: aws/managed-test@v1
    Inputs:
      Sources:
        - WorkflowSource
    Outputs:
      Reports:
        CoverageReport:
```

```
          Format: CLOVERXML
          IncludePaths:
            - "coverage/*"
        TestReport:
          Format: JUNITXML
          IncludePaths:
            - junit.xml
    Configuration:
      Steps:
        - Run: npm install
        - Run: npm run test
  BuildBackend:
    Identifier: aws/build@v1
    DependsOn:
      - Test
    Environment:
      Name: codecatalyst-cfn-environment
      Connections:
        - Name: codecatalyst-account-connection
          Role: codecatalyst-build-role
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      Steps:
        - Run: . ./setup-sam.sh
        - Run: sam package --template-file sam-template.yml --s3-
bucket codecatalyst-cfn-s3-bucket --output-template-file sam-template-packaged.yml
 --region us-west-2
    Outputs:
      Artifacts:
        - Name: buildArtifact
          Files:
            - "**/*"
  DeployCloudFormationStack:
    Identifier: aws/cfn-deploy@v1
    DependsOn:
      - BuildBackend
    Environment:
      Name: codecatalyst-cfn-environment
      Connections:
        - Name: codecatalyst-account-connection
          Role: codecatalyst-deploy-role
    Inputs:
```

```
    Artifacts:
      - buildArtifact
    Sources: []
  Configuration:
    name: codecatalyst-cfn-stack
    region: us-west-2
    role-arn: arn:aws:iam::111122223333:role/StackRole
    template: ./sam-template-packaged.yml
    capabilities: CAPABILITY_IAM,CAPABILITY_AUTO_EXPAND
```

上記のコードで、以下を置き換えます。

- 環境の名前*codecatalyst-cfn-environment*を持つ の両方のインスタンス。
- アカウント接続の表示名*codecatalyst-account-connection*を持つ の両方のインスタンス。表示名は数字にすることができます。詳細については、「ステップ 3: に AWS ロールを追加する CodeCatalyst」を参照してください。
- *codecatalyst-build-role* に、 で作成したビルドロールの名前を入力しますステップ 2: AWS ロールを作成する。
- で作成した Amazon *codecatalyst-cfn-sS3 ####*の名前を持つ 3 バケットステップ 4: Amazon S3 バケットを作成する。 Amazon S3
- Amazon S3 バケットが存在するリージョン (最初のインスタンス) とスタックがデプロイされるリージョン (2 番目のインスタンス) を持つ *us-west-2* のインスタンス。これらのリージョンは異なる場合があります。このチュートリアルでは、両方のリージョンが に設定されていることを前提としていますus-west-2。Amazon S3 および でサポートされているリージョンの詳細については AWS CloudFormation、「」の「サービスエンドポイントとクォータ」を参照してくださいAWS 全般のリファレンス。
- *codecatalyst-deploy-role* で作成したデプロイロールの名前。 ステップ 2: AWS ロールを作成する
- *codecatalyst-cfn-environment* で作成した環境の名前前提条件。
- *arn:aws:iam::111122223333:role/StackRole* で作成したスタックロールの Amazon リソースネーム (ARN)ステップ 2: AWS ロールを作成する。

> **Note**
>
> ビルド、デプロイ、スタックロールを作成しない場合は、*codecatalyst-build-role*、、*codecatalyst-deploy-role*および *arn:aws:iam::111122223333:role/StackRole*

をCodeCatalystWorkflowDevelopmentRole-*spaceName*ロールの名前または ARN に置き換えます。このロールの詳細については、「ステップ 2: AWS ロールを作成する」を参照してください。

前述のコードのプロパティについては、「」を参照してくださいAWS CloudFormation 「スタックのデプロイ」アクション YAML 定義。

8. (オプション) 検証 を選択して、コミットする前に YAML コードが有効であることを確認します。
9. [Commit] (コミット) を選択します。
10. コミットワークフローダイアログボックスで、次のように入力します。

    a. ワークフローファイル名 の場合、デフォルト のままにしますcodecatalyst-cfn-workflow。

    b. コミットメッセージ には、次のように入力します。

    ```
    add initial workflow file
    ```

    c. リポジトリ で、 を選択しますcodecatalyst-cfn-source-repository。

    d. ブランチ名 で、メイン を選択します。

    e. [Commit] (コミット) を選択します。

これでワークフローが作成されました。ワークフロー実行は、ワークフローの上部で定義されたトリガーが原因で自動的に開始されます。具体的には、codecatalyst-cfn-workflow.yamlファイルをソースリポジトリにコミット (およびプッシュ) すると、トリガーによってワークフロー実行が開始されます。

進行中のワークフロー実行を表示するには

1. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
2. 先ほど作成したワークフローを選択します: codecatalyst-cfn-workflow。
3. 実行 タブを選択します。
4. Run ID 列で、実行 ID を選択します。
5. テストを選択して、テストの進行状況を確認します。
6. ビルドの進行状況BuildBackendを表示するには、 を選択します。

7. DeployCloudFormationStack を選択して、デプロイの進行状況を確認します。

   実行の詳細の表示の詳細については、「」を参照してくださいワークフローの実行ステータスと詳細の表示。

8. DeployCloudFormationStack アクションが終了したら、次の操作を行います。

   - ワークフローの実行が成功したら、次の手順に進みます。
   - テストまたはBuildBackendアクションでワークフローの実行が失敗した場合は、ログを選択して問題をトラブルシューティングします。
   - ワークフローの実行が DeployCloudFormationStackアクションで失敗した場合は、デプロイアクションを選択し、概要タブを選択します。CloudFormation イベントセクションまでスクロールして、詳細なエラーメッセージを表示します。ロールバックが発生した場合は、ワークフローを再実行する前に、 の AWS CloudFormation AWS コンソールからcodecatalyst-cfn-stackスタックを削除します。

デプロイを確認するには

1. デプロイが成功したら、上部の横のメニューバーから可変 (7) を選択します。 (右側のペインでは変数を選択しないでください) 。
2. の横にある `https://` URL をブラウザにHelloWorldApi貼り付けます。

   Lambda 関数からの hello world JSON メッセージが表示され、ワークフローが Lambda 関数と API Gateway を正常にデプロイして設定したことを示します。

   > **Tip**
   >
   > この URL をワークフロー図にいくつかの小さな設定で CodeCatalyst 表示できます。詳細については、「ワークフロー図にデプロイされたアプリケーションの URL を表示する」を参照してください。

ユニットテスト結果とコードカバレッジを検証するには

1. ワークフロー図で、テスト を選択し、レポート を選択します。
2. ユニットテスト結果TestReportを表示するか、テスト対象のファイルのコードカバレッジの詳細を表示するCoverageReportか、この場合は `app.js`および を選択しますtest-handler.js。

デプロイされたリソースを確認するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/apigateway/ で API Gateway コンソールを開きます。
2. AWS SAM テンプレートが作成した codecatalyst-cfn-stack API を確認します。API 名は、ワークフロー定義ファイル () `Configuration/name`の値から取得されます`codecatalyst-cfn-workflow.yaml`。
3. https://console.aws.amazon.com/lambda/ で AWS Lambda コンソールを開きます。
4. ナビゲーションペインで、[関数] を選択します。
5. Lambda 関数 を選択します`codecatalyst-cfn-stack-HelloWorldFunction-`*`string`*。
6. API Gateway が関数のトリガーであることを確認できます。この統合は、 AWS SAM `AWS::Serverless::Function` リソースタイプによって自動的に設定されました。

ステップ 7: 変更を行う

このステップでは、Lambda ソースコードを変更してコミットします。このコミットにより、新しいワークフロー実行が開始されます。この実行では、Lambda コンソールで指定されたデフォルトのトラフィックシフト設定を使用するブルー/グリーンスキームで新しい Lambda 関数をデプロイします。

Lambda ソースを変更するには

1. で CodeCatalyst、プロジェクトに移動します。
2. ナビゲーションペインで [コード] を選択してから、[ソースリポジトリ] を選択します。
3. ソースリポジトリ を選択します`codecatalyst-cfn-source-repository`。
4. アプリケーションファイルを変更します。
   a. `hello-world` フォルダを選択します。
   b. `app.js` ファイルを選択します。
   c. [編集] を選択します。
   d. 23 行目で、 `hello world`を に変更します**`Tutorial complete!`**。
   e. コミット を選択し、もう一度コミット を選択します。

      コミットにより、ワークフローの実行が開始されます。ユニットテストを更新して名前の変更を反映していないため、この実行は失敗します。

5. ユニットテストを更新します。

   a. [hello-world\tests\unit\test-handler.js] を選択します。

   b. [編集] を選択します。

   c. 19 行目で、 hello worldを に変更しますTutorial complete!。

   d. コミット を選択し、もう一度コミット を選択します。

      コミットにより、別のワークフロー実行が開始されます。この実行は成功します。

6. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
7. を選択しcodecatalyst-cfn-workflow、実行 を選択します。
8. 最新の実行の実行 ID を選択します。まだ進行中である必要があります。
9. テスト 、BuildBackend、および を選択してDeployCloudFormationStack、ワークフロー実行の進行状況を確認します。
10. ワークフローが終了したら、上部にある変数 (7) を選択します。
11. の横にある https:// URL をブラウザにHelloWorldApi貼り付けます。

    ブラウザに、新しいアプリケーションが正常にデプロイされたことを示すTutorial complete!メッセージが表示されます。

クリーンアップ

このチュートリアルで使用するファイルとサービスをクリーンアップして、料金が発生しないようにします。

CodeCatalyst コンソールでクリーンアップするには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. を削除しますcodecatalyst-cfn-workflow。
3. を削除しますcodecatalyst-cfn-environment。
4. を削除しますcodecatalyst-cfn-source-repository。
5. を削除しますcodecatalyst-cfn-project。

でクリーンアップするには AWS Management Console

1. 次のように CloudFormation、 でクリーンアップします。

a. https://console.aws.amazon.com/cloudformation で AWS CloudFormation コンソールを開きます。

b. `codecatalyst-cfn-stack` を削除します。

スタックを削除すると、API Gateway および Lambda サービスからすべてのチュートリアルリソースが削除されます。

2. Amazon S3 で次のようにクリーンアップします。

a. https://console.aws.amazon.com/s3/でAmazon S3 コンソールを開きます。

b. [`codecatalyst-cfn-s3-bucket`] を選択します。

c. バケットの内容を削除します。

d. バケットを削除します。

3. IAM で次のようにクリーンアップします。

a. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。

b. `codecatalyst-deploy-policy` を削除します。

c. `codecatalyst-build-policy` を削除します。

d. `codecatalyst-stack-policy` を削除します。

e. `codecatalyst-deploy-role` を削除します。

f. `codecatalyst-build-role` を削除します。

g. `codecatalyst-stack-role` を削除します。

このチュートリアルでは、 CodeCatalyst ワークフローとスタックのデプロイアクションを使用してサーバーレスアプリケーションを CloudFormation スタックとしてデプロイする方法を学習しました。 AWS CloudFormation

AWS CloudFormation 「スタックのデプロイ」アクションの追加

次の手順を使用して、 AWS CloudFormation スタックのデプロイアクションをワークフローに追加します。

Visual

ビジュアルエディタを使用して AWS CloudFormation 「スタックのデプロイ」アクションを追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. ビジュアル を選択します。
7. 左上で、+ Actions を選択してアクションカタログを開きます。
8. ドロップダウンリストから Amazon CodeCatalystを選択します。
9. AWS CloudFormation スタックのデプロイアクションを検索し、次のいずれかを実行します。

   - プラス記号 (+) を選択してワークフロー図にアクションを追加し、その設定ペインを開きます。

     または

   - AWS CloudFormation スタックのデプロイ を選択します。アクションの詳細ダイアログボックスが表示されます。このダイアログボックスで、次の操作を行います。
     - (オプション) ダウンロード を選択して、アクションのソースコード を表示します。
     - ワークフローに追加 を選択して、ワークフロー図にアクションを追加し、その設定ペインを開きます。
10. 入力タブと設定タブで、必要に応じてフィールドに入力します。各フィールドの説明については、「」を参照してくださいAWS CloudFormation 「スタックのデプロイ」アクション YAML 定義。このリファレンスでは、YAML エディタとビジュアルエディタの両方に表示される各フィールド (および対応する YAML プロパティ値) に関する詳細情報を提供します。
11. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
12. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

YAML

YAML エディタを使用して AWS CloudFormation 「スタックのデプロイ」アクションを追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. YAML を選択します。
7. 左上で + Actions を選択してアクションカタログを開きます。
8. ドロップダウンリストから、Amazon CodeCatalystを選択します。
9. AWS CloudFormation スタックのデプロイアクションを検索し、次のいずれかを実行します。

   - プラス記号 (+) を選択してワークフロー図にアクションを追加し、設定ペインを開きます。

     または

   - AWS CloudFormation スタック をデプロイ を選択します。アクションの詳細ダイアログボックスが表示されます。このダイアログボックスで、次の操作を行います。
     - (オプション) ダウンロード を選択して、アクションのソースコード を表示します。
     - ワークフローに追加 を選択して、ワークフロー図にアクションを追加し、その設定ペインを開きます。
10. 必要に応じて YAML コードのプロパティを変更します。使用可能な各プロパティの説明は、「」に記載されていますAWS CloudFormation 「スタックのデプロイ」アクション YAML 定義。
11. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
12. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

ロールバックの設定

デフォルトでは、 AWS CloudFormation スタックのデプロイアクションが失敗すると、 はスタック AWS CloudFormation を最後の既知の安定状態にロールバックします。アクションが失敗したときだけでなく、指定された Amazon CloudWatch アラームが発生したときにもロールバックが発生するように動作を変更できます。 CloudWatch アラームの詳細については、「Amazon ユーザーガイド」の「Amazon CloudWatch アラームの使用」を参照してください。 CloudWatch

アクションが失敗したときに CloudFormation がスタックをロールバックしないように、デフォルトの動作を変更することもできます。

ロールバックを設定するには、以下の手順に従います。

> **Note**
>
> ロールバックを手動で開始することはできません。

Visual

開始する前に

1. 機能している AWS CloudFormation スタックのデプロイアクションを含むワークフローがあることを確認してください。詳細については、「ワークフローを使用した AWS CloudFormation スタックのデプロイ」を参照してください。
2. スタックのデプロイアクションの スタックロール - オプションフィールドで指定されたロールには、 アクセスCloudWatchFullAccess許可を必ず含めてください。 AWS CloudFormation 適切なアクセス許可を持つこのロールの作成については、「」を参照してくださいステップ 2: AWS ロールを作成する。

AWS CloudFormation 「スタックのデプロイ」アクションのロールバックアラームを設定するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。

4. スタックのデプロイ AWS CloudFormation アクションを含むワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. ビジュアル を選択します。
7. AWS CloudFormation スタックのデプロイアクションを選択します。
8. 詳細ペインで、設定 を選択します。
9. 下部で、アドバンスト を展開します。
10. Monitor alarm ARNs で、Add alarm を選択します。
11. 以下のフィールドに情報を入力します。

    - アラーム ARN

      ロールバックトリガーとして使用する Amazon CloudWatch アラームの Amazon リソースネーム (ARN) を指定します。例えば `arn:aws:cloudwatch::123456789012:alarm/MyAlarm` です。最大 5 つのロールバックトリガーを設定できます。

      > **Note**
      >
      > CloudWatch アラーム ARN を指定する場合は、アクションが にアクセスできるようにする追加のアクセス許可も設定する必要があります CloudWatch。詳細については、「ロールバックの設定」を参照してください。

    - モニタリング時間

      が指定されたアラームを CloudFormation モニタリングする 0 ~ 180 分までの時間を指定します。モニタリングは、すべてのスタックリソースがデプロイされた後に開始されます。アラームが指定されたモニタリング時間内に発生した場合、デプロイは失敗し、スタックオペレーション全体を CloudFormation ロールバックします。

      デフォルト: 0. は、スタックリソースのデプロイ中、後ではなく、アラーム CloudFormation のみをモニタリングします。

YAML

AWS CloudFormation 「スタックのデプロイ」アクションのロールバックトリガーを設定するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. スタックのデプロイ AWS CloudFormation アクションを含むワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. YAML を選択します。
7. YAML コードに `monitor-alarm-arns`および `monitor-timeout-in-minutes`プロパティを追加して、ロールバックトリガーを追加します。各プロパティの説明については、「」を参照してくださいAWS CloudFormation 「スタックのデプロイ」アクション YAML 定義。
8. AWS CloudFormation スタックのデプロイアクションの `role-arn`プロパティで指定されたロールには、 アクセスCloudWatchFullAccess許可を必ず含めてください。適切なアクセス許可を持つこのロールの作成については、「」を参照してくださいステップ 2: AWS ロールを作成する。

Visual

AWS CloudFormation 「スタックのデプロイ」アクションのロールバックを無効にするには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. スタックのデプロイ AWS CloudFormation アクションを含むワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. ビジュアル を選択します。

7. AWS CloudFormation スタックのデプロイアクションを選択します。
8. 詳細ペインで、設定 を選択します。
9. 下部で、アドバンスト を展開します。
10. ロールバックの無効化 をオンにします。

YAML

AWS CloudFormation 「スタックのデプロイ」アクションのロールバックを無効にするには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. スタックのデプロイ AWS CloudFormation アクションを含むワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. YAML を選択します。
7. ロールバックを停止するには、YAML コードに `disable-rollback: 1`プロパティを追加します。このプロパティの説明については、「」を参照してくださいAWS CloudFormation 「スタックのデプロイ」アクション YAML 定義。

AWS CloudFormation 「スタックのデプロイ」アクションによって生成される変数

AWS CloudFormation スタックのデプロイアクションは、実行時に次の変数を生成して設定します。これらは事前定義された変数 と呼ばれます。

ワークフローでこれらの変数を参照する方法については、「」を参照してください事前定義された変数の使用。

キー	値
デプロイプラットフォーム	デプロイプラットフォームの名前。 にハードコードされますAWS:CloudFormation 。

キー	値
region	ワークフローの実行中に にデプロイ AWS リージョン された のリージョンコード。 例: `us-west-2`
スタック ID	デプロイされたスタックの Amazon リソースネーム (ARN)。 例: `arn:aws:cloudformation:us-west-2:111122223333:stack/codecatalyst-cfn-stack/6aad4380-100a-11ec-a10a-03b8a84d40df`

AWS CloudFormation 「スタックのデプロイ」アクション YAML 定義

AWS CloudFormation スタックのデプロイアクションの YAML 定義を次に示します。このアクションの使用方法については、「」を参照してくださいワークフローを使用した AWS CloudFormation スタックのデプロイ。

このアクション定義は、より広範なワークフロー定義ファイル内のセクションとして存在します。ファイルの詳細については、「ワークフロー YAML 定義」を参照してください。

> **Note**
>
> 以下の YAML プロパティのほとんどには、対応する UI 要素がビジュアルエディタにあります。UI 要素を検索するには、Ctrl+F を使用します。 要素は、関連付けられた YAML プロパティとともに一覧表示されます。

```
# The workflow definition starts here.
# See ######### for details.

Name: MyWorkflow
SchemaVersion: 1.0
Actions:

# The action definition starts here.
```

```
  DeployCloudFormationStack:
    Identifier: aws/cfn-deploy@v1
    DependsOn:
      - build-action
    Compute:
      Type: EC2 | Lambda
      Fleet: fleet-name
    Timeout: timeout-minutes
    Environment:
      Name: environment-name
      Connections:
        - Name: account-connection-name
          Role: DeployRole
    Inputs:
      Sources:
        - source-name-1
      Artifacts:
        - CloudFormation-artifact
    Configuration:
      name: stack-name
      region: us-west-2
      template: template-path
      role-arn: arn:aws:iam::123456789012:role/StackRole
      capabilities: CAPABILITY_IAM,CAPABILITY_NAMED_IAM,CAPABILITY_AUTO_EXPAND
      parameter-overrides: KeyOne=ValueOne,KeyTwo=ValueTwo | path-to-JSON-file
      no-execute-changeset: 1|0
      fail-on-empty-changeset: 1|0
      disable-rollback: 1|0
      termination-protection: 1|0
      timeout-in-minutes: minutes
      notification-arns: arn:aws:sns:us-east-1:123456789012:MyTopic,arn:aws:sns:us-
east-1:123456789012:MyOtherTopic
      monitor-alarm-arns: arn:aws:cloudwatch::123456789012:alarm/
MyAlarm,arn:aws:cloudwatch::123456789012:alarm/MyOtherAlarm
      monitor-timeout-in-minutes: minutes
      tags: '[{"Key":"MyKey1","Value":"MyValue1"},{"Key":"MyKey2","Value":"MyValue2"}]'
```

DeployCloudFormationStack

(必須)

アクションの名前を指定します。すべてのアクション名は、ワークフロー内で一意である必要があります。アクション名は、英数字 (a～z、A～Z、0～9)、ハイフン (-)、アンダースコア (_) に制限され

ています。スペースは使用できません。引用符を使用してアクション名で特殊文字やスペースを有効にすることはできません。

デフォルト: `DeployCloudFormationStack_nn`。

対応する UI: 設定タブ/アクションの表示名

Identifier

(*DeployCloudFormationStack*/Identifier)

(必須)

アクションを識別します。バージョンを変更しない限り、このプロパティを変更しないでください。詳細については、「アクションのメジャー、マイナー、またはパッチバージョンの指定」を参照してください。

デフォルト: `aws/cfn-deploy@v1`。

対応する UI: ワークフロー図/DeployCloudFormationStack_nn/aws/cfn-deploy@v1 ラベル

DependsOn

(*DeployCloudFormationStack*/DependsOn)

(オプション)

このアクションを実行するために正常に実行する必要があるアクション、アクショングループ、またはゲートを指定します。

「依存」機能の詳細については、「」を参照してください他のアクションに依存するようにアクションを設定する。

対応する UI: の入力タブ/依存 - オプション

Compute

(*DeployCloudFormationStack*/Compute)

(オプション)

ワークフローアクションを実行するために使用されるコンピューティングエンジン。コンピューティングはワークフローレベルまたはアクションレベルで指定できますが、両方を指定することはできません。ワークフローレベルで指定すると、コンピューティング設定はワークフローで定義されたすべ

てのアクションに適用されます。ワークフローレベルでは、同じインスタンスで複数のアクションを実行することもできます。詳細については、「アクション間でのコンピューティングの共有」を参照してください。

対応する UI: なし

Type

(*DeployCloudFormationStack*/Compute/Type)

(Computeが含まれている場合は必須)

コンピューティングエンジンのタイプ。次のいずれかの値を使用できます。

- EC2 (ビジュアルエディタ) または EC2 (YAML エディタ)

  アクション実行中の柔軟性のために最適化されました。
- Lambda (ビジュアルエディタ) または Lambda (YAML エディタ)

  アクションの起動速度を最適化しました。

コンピューティングタイプの詳細については、「コンピューティングタイプ」を参照してください。

対応する UI: 設定タブ/詳細 - オプション/コンピューティングタイプ

Fleet

(*DeployCloudFormationStack*/Compute/Fleet)

(オプション)

ワークフローまたはワークフローアクションを実行するマシンまたはフリートを指定します。オンデマンドフリートでは、アクションが開始されると、ワークフローは必要なリソースをプロビジョニングし、アクションが終了するとマシンは破棄されます。オンデマンドフリートの例: `Linux.x86-64.Large`、`Linux.x86-64.XLarge`。オンデマンドフリートの詳細については、「」を参照してくださいオンデマンドフリートのプロパティ。

プロビジョニングされたフリートでは、ワークフローアクションを実行するように一連の専用マシンを設定します。これらのマシンはアイドル状態のままで、アクションをすぐに処理できます。プロビジョニングされたフリートの詳細については、「」を参照してくださいプロビジョニングされたフリートのプロパティ。

`Fleet` を省略した場合、デフォルトは ですLinux.x86-64.Large。

対応する UI: 設定タブ/アドバンスト - オプション/コンピューティングフリート

Timeout

(*DeployCloudFormationStack*/Timeout)

(オプション)

がアクション CodeCatalyst を終了するまでにアクションを実行できる時間を分単位で指定します (YAML エディタ)、または時間と分単位で指定します (ビジュアルエディタ)。最小値は 5 分で、最大値は「」で説明されていますワークフローのクォータ。デフォルトのタイムアウトは、最大タイムアウトと同じです。

対応する UI: 設定タブ/タイムアウトを分単位で - オプション

Environment

(*DeployCloudFormationStack*/Environment)

(必須)

アクションで使用する CodeCatalyst 環境を指定します。

環境の詳細については、環境を使用して AWS アカウント および VPCsにデプロイする CodeCatalyst 「」および「」を参照してください環境を作成する。

対応する UI: 設定タブ/'環境/アカウント/ロール'/環境

Name

(*DeployCloudFormationStack*/Environment/Name)

(Environmentが含まれている場合は必須)

アクションに関連付ける既存の環境の名前を指定します。

対応する UI: 設定タブ/'環境/アカウント/ロール'/環境

Connections

(*DeployCloudFormationStack*/Environment/Connections)

(Environmentが含まれている場合は必須)

アクションに関連付けるアカウント接続を指定します。で最大 1 つのアカウント接続を指定できますEnvironment。

アカウント接続の詳細については、「」を参照してください接続された AWS リソースへのアクセスを許可する AWS アカウント。アカウント接続を環境に関連付ける方法については、「」を参照してください環境を作成する。

対応する UI: 設定タブ/'環境/アカウント/ロール'/AWS アカウント接続

Name

(*DeployCloudFormationStack*/Environment/Connections/Name)

(必須)

アカウント接続の名前を指定します。

対応する UI: 設定タブ/'環境/アカウント/ロール'/AWS アカウント接続

Role

(*DeployCloudFormationStack*/Environment/Connections/Role)

(必須)

AWS CloudFormation スタックのデプロイアクションが および サービスにアクセスするために使用する IAM ロールの名前を指定します。 AWS AWS CloudFormation

このロールに次のポリシーが含まれていることを確認します。

- 次のアクセス許可ポリシー :

> **Warning**
>
> アクセス許可は、次のポリシーに示すものに制限します。より広範なアクセス許可を持つロールを使用すると、セキュリティ上のリスクが生じる可能性があります。

```
{
    "Version": "2012-10-17",
    "Statement": [{
    "Action": [
        "cloudformation:CreateStack",
```

```
        "cloudformation:DeleteStack",
        "cloudformation:Describe*",
        "cloudformation:UpdateStack",
        "cloudformation:CreateChangeSet",
        "cloudformation:DeleteChangeSet",
        "cloudformation:ExecuteChangeSet",
        "cloudformation:SetStackPolicy",
        "cloudformation:ValidateTemplate",
        "cloudformation:List*",
        "iam:PassRole"
    ],
    "Resource": "*",
    "Effect": "Allow"
}]
}
```

> **Note**
>
> ロールを初めて使用する場合は、リソースポリシーステートメントで次のワイルドカードを使用し、使用可能になった後にリソース名でポリシーの範囲を絞り込みます。
>
> ```
> "Resource": "*"
> ```

- 次のカスタム信頼ポリシー：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                 ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

このロールがスペース内のアカウント接続に追加されていることを確認します。アカウント接続への IAM ロールの追加の詳細については、「」を参照してくださいアカウント接続への IAM ロールの追加。

> **Note**
>
> 必要に応じて、ここでCodeCatalystWorkflowDevelopmentRole-*spaceName*ロールの名前を指定できます。このロールの詳細については、「アカウントとスペースのCodeCatalystWorkflowDevelopmentRole-***spaceName***ロールの作成」を参照してください。CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールには、セキュリティリスクをもたらす可能性のある非常に広範なアクセス許可があることを理解します。このロールは、セキュリティが懸念されないチュートリアルとシナリオでのみ使用することをお勧めします。

対応する UI: 設定タブ/'環境/アカウント/ロール'/ロール

Inputs

(*DeployCloudFormationStack*/Inputs)

(オプション)

Inputs セクションでは、ワークフローの実行中に がDeployCloudFormationStack必要とするデータを定義します。

> **Note**
>
> AWS CloudFormation スタックのデプロイアクションごとに最大 4 つの入力 (1 つのソースと 3 つのアーティファクト) が許可されます。

異なる入力 (ソースとアーティファクトなど) にあるファイルを参照する必要がある場合、ソース入力はプライマリ入力、アーティファクトはセカンダリ入力です。セカンダリ入力内のファイルへの参照には、プライマリからファイルを分散するための特別なプレフィックスが付けられます。詳細については、「例: 複数のアーティファクト内のファイルを参照する」を参照してください。

対応する UI: Inputs タブ

Sources

(*DeployCloudFormationStack*/Inputs/Sources)

(CloudFormation または AWS SAM テンプレートがソースリポジトリに保存されている場合に必須)

CloudFormation または AWS SAM テンプレートがソースリポジトリに保存されている場合は、そのソースリポジトリのラベルを指定します。現在、サポートされているラベルは のみですWorkflowSource。

CloudFormation または AWS SAM テンプレートがソースリポジトリに含まれていない場合は、別のアクションによって生成されたアーティファクト、または Amazon S3 バケットに存在する必要があります。

sources の詳細については、「ワークフローをソースリポジトリに接続する」を参照してください。

対応する UI: 入力タブ/ソース - オプション

Artifacts - input

(*DeployCloudFormationStack*/Inputs/Artifacts)

(CloudFormation または AWS SAM テンプレートが前のアクションの出力アーティファクトに保存されている場合に必要です)

デプロイする CloudFormation または AWS SAM テンプレートが、前のアクションによって生成されたアーティファクトに含まれている場合は、ここでそのアーティファクトを指定します。CloudFormation テンプレートがアーティファクトに含まれていない場合は、ソースリポジトリまたは Amazon S3 バケットに存在する必要があります。

アーティファクトの例などの詳細については、「」を参照してくださいアーティファクトを使用したワークフロー内のアクション間のデータの共有。

対応する UI: 設定タブ/アーティファクト - オプション

Configuration

(*DeployCloudFormationStack*/Configuration)

(必須)

アクションの設定プロパティを定義できるセクション。

対応する UI: 設定タブ

name

(*DeployCloudFormationStack*/Configuration/name)

(必須)

CloudFormation スタックのデプロイ AWS CloudFormation アクションが作成または更新するスタックの名前を指定します。

対応する UI: 設定タブ/スタック名

region

(*DeployCloudFormationStack*/Configuration/region)

(必須)

スタックをデプロイする AWS リージョン を指定します。リージョンコードの一覧については、「リージョンエンドポイント」を参照してください。

対応する UI: 設定タブ/スタックリージョン

template

(*DeployCloudFormationStack*/Configuration/template)

(必須)

CloudFormation または AWS SAM テンプレートファイルの名前とパスを指定します。テンプレートは JSON または YAML 形式で、ソースリポジトリ、以前のアクションのアーティファクト、または Amazon S3 バケットに格納できます。テンプレートファイルがソースリポジトリまたはアーティファクトにある場合、パスはソースまたはアーティファクトのルートを基準にしています。テンプレートが Amazon S3 バケットにある場合、パスはテンプレートのオブジェクト URL 値です。

例:

```
./MyFolder/MyTemplate.json
```

```
MyFolder/MyTemplate.yml
```

```
https://MyBucket.s3.us-west-2.amazonaws.com/MyTemplate.yml
```

> **Note**
>
> テンプレートのファイルパスにプレフィックスを追加して、見つけるアーティファクトまたはソースを示す必要がある場合があります。詳細については、「ソースリポジトリ内のファイルを参照する」および「アーティファクト内のファイルを参照する」を参照してください。

対応する UI: 設定タブ/テンプレート

role-arn

(*DeployCloudFormationStack*/Configuration/role-arn)

(必須)

スタック role の Amazon リソースネーム (ARN) を指定します。 CloudFormation は、このロールを使用してスタック内のリソースにアクセスし、変更します。例: `arn:aws:iam::123456789012:role/StackRole`。

スタックロールに以下が含まれていることを確認します。

- 1 つ以上のアクセス許可ポリシー。ポリシーは、スタック内のリソースによって異なります。例えば、スタックに AWS Lambda 関数が含まれている場合は、Lambda へのアクセスを許可するアクセス許可を追加する必要があります。「」で説明スタックロールを作成するにはされているチュートリアルに従った場合チュートリアル: を使用してサーバーレスアプリケーションをデプロイする AWS CloudFormation、 というタイトルの手順が含まれており、一般的なサーバーレスアプリケーションスタックをデプロイする場合にスタックロールが必要とするアクセス許可が一覧表示されます。

  > **Warning**
  >
  > スタック内のリソースにアクセスするために CloudFormation サービスが必要とするアクセス許可に制限します。より広範なアクセス許可を持つロールを使用すると、セキュリティ上のリスクが生じる可能性があります。

- 次の信頼ポリシー :

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": "cloudformation.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

必要に応じて、このロールをアカウント接続に関連付けます。IAM ロールとアカウント接続の関連付けの詳細については、「」を参照してくださいアカウント接続への IAM ロールの追加。スタックロールをアカウント接続に関連付けない場合、スタックロールはビジュアルエディタのスタックロールのドロップダウンリストに表示されません。ただし、ロール ARN は YAML エディタを使用してrole-arnフィールドで指定できます。

> **Note**
>
> 必要に応じて、ここでCodeCatalystWorkflowDevelopmentRole-*spaceName*ロールの名前を指定できます。このロールの詳細については、「アカウントとスペースのCodeCatalystWorkflowDevelopmentRole-***spaceName***ロールの作成」を参照してください。CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールには、セキュリティリスクをもたらす可能性のある非常に広範なアクセス許可があることを理解します。このロールは、セキュリティが懸念されないチュートリアルとシナリオでのみ使用することをお勧めします。

対応する UI: 設定タブ/スタックロール - オプション

capabilities

(*DeployCloudFormationStack*/Configuration/capabilities)

(必須)

が特定のスタックを作成 AWS CloudFormation するために必要な IAM 機能のリストを指定します。ほとんどの場合、 をデフォルト値の capabilitiesのままにしておくことができますCAPABILITY_IAM,CAPABILITY_NAMED_IAM,CAPABILITY_AUTO_EXPAND。

スタックのデプロイ AWS CloudFormation アクションのログ`##[error] requires capabilities: [capability-name]`に が表示されている場合は、IAM 機能エラーを修正するにはどうすればよいですか？「」を参照して問題を解決する方法を確認してください。

IAM 機能の詳細については、「IAM ユーザーガイド」の AWS CloudFormation 「 テンプレートでの IAM リソースの承認」を参照してください。

対応する UI: 設定タブ/アドバンスト/機能

parameter-overrides

(*DeployCloudFormationStack*/Configuration/parameter-overrides)

(オプション)

デフォルト値を持たない AWS CloudFormation 、またはデフォルト以外の値を指定するパラメータを または AWS SAM テンプレートに指定します。パラメータの詳細については、「 ユーザーガイド」の「パラメータ AWS CloudFormation 」を参照してください。

`parameter-overrides` プロパティは以下を受け入れます。

- パラメータと値を含む JSON ファイル。
- パラメータと値のカンマ区切りリスト。

JSON ファイルを指定するには

1. JSON ファイルが次のいずれかの構文を使用していることを確認します。

```
{
  "Parameters": {
    "Param1": "Value1",
    "Param2": "Value2",
    ...
  }
}
```

または...

```
[
  {
     "ParameterKey": "Param1",
```

```
    "ParameterValue": "Value1"
  },
  ...
]
```

（他の構文もありますが、書き込み CodeCatalyst 時には ではサポートされていません）。JSON ファイルで CloudFormation パラメータを指定する方法の詳細については、 コマンドAWS CLI リファレンスの「サポートされている JSON 構文」を参照してください。

2. 次のいずれかの形式を使用して、JSON ファイルへのパスを指定します。

   - JSON ファイルが前のアクションの出力アーティファクトにある場合は、以下を使用します。

     `file:///artifacts/`*`current-action-name`*`/`*`output-artifact-name`*`/`*`path-to-json-file`*

     詳細については、例 1 を参照してください。

   - JSON ファイルがソースリポジトリにある場合は、以下を使用します。

     `file:///sources/WorkflowSource/`*`path-to-json-file`*

     詳細については、例 2 を参照してください。

     例 1 – JSON ファイルは出力アーティファクトにあります

```
##My workflow YAML
...
Actions:
  MyBuildAction:
    Identifier: aws/build@v1
    Outputs:
      Artifacts:
        - Name: ParamArtifact
          Files:
            - params.json
    Configuration:
    ...
  MyDeployCFNStackAction:
    Identifier: aws/cfn-deploy@v1
    Configuration:
      parameter-overrides: file:///artifacts/MyDeployCFNStackAction/
ParamArtifact/params.json
```

例 2 – JSON ファイルは、ソースリポジトリの という名前のフォルダにあります。 `my/folder`

```
##My workflow YAML
...
Actions:
  MyDeployCloudFormationStack:
    Identifier: aws/cfn-deploy@v1
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      parameter-overrides: file:///sources/WorkflowSource/my/folder/params.json
```

パラメータのカンマ区切りリストを使用するには

- 次の形式を使用して、 `parameter-overrides` プロパティにパラメータの名前と値のペアを追加します。

  param-1=*value-1*,*param-2*=*value-2*

  例えば、次の AWS CloudFormation テンプレートを想定します。

```
##My CloudFormation template

Description: My AWS CloudFormation template

Parameters:
  InstanceType:
    Description: Defines the Amazon EC2 compute for the production server.
    Type: String
    Default: t2.micro
    AllowedValues:
      - t2.micro
      - t2.small
      - t3.medium

Resources:
...
```

プロパティはparameter-overrides次のように設定できます。

```
##My workflow YAML
...
Actions:
...
  DeployCloudFormationStack:
    Identifier: aws/cfn-deploy@v1
    Configuration:
      parameter-overrides: InstanceType=t3.medium,UseVPC=true
```

> **Note**
>
> を値undefinedとして使用して、対応する値なしでパラメータ名を指定できます。例: `parameter-overrides: MyParameter=undefined`
> その結果、スタックの更新中に、 は指定されたパラメータ名に既存のパラメータ値 CloudFormation を使用します。

対応する UI:

- 設定タブ/詳細/パラメータの上書き
- ファイルを使用して設定タブ/詳細/パラメータオーバーライド/オーバーライドを指定する
- 設定タブ/詳細/パラメータオーバーライド/値セットを使用したオーバーライドの指定

no-execute-changeset

(*DeployCloudFormationStack*/Configuration/no-execute-changeset)

(オプション)

CloudFormation 変更セット CodeCatalyst を作成するかどうかを指定し、実行する前に停止します。これにより、 CloudFormation コンソールで変更セットを確認できます。変更セットが正常に見えると判断した場合は、このオプションを無効にしてからワークフローを再実行し、 が停止せずに変更セットを作成して実行 CodeCatalyst できるようにします。デフォルトでは、停止せずに変更セットを作成して実行します。詳細については、「 コマンドリファレンス」の AWS CloudFormation 「デプロイパラメータ」を参照してください。 AWS CLI 変更セットの表示の詳細については、「 ユーザーガイド」の「変更セットの表示」を参照してください。 AWS CloudFormation

対応する UI: 設定タブ/詳細/実行なしの変更セット

fail-on-empty-changeset

(*DeployCloudFormationStack*/Configuration/fail-on-empty-changeset)

(オプション)

CloudFormation 変更セットが空の場合、 AWS CloudFormation スタックのデプロイアクションを失敗 CodeCatalyst させるかどうかを指定します。 (変更セットが空の場合、最新のデプロイ中にスタックに変更が加えられなかったことを意味します)。デフォルトでは、変更セットが空の場合にアクションを続行し、スタックが更新されていなくてもUPDATE_COMPLETEメッセージを返すことができます。

この設定の詳細については、「 コマンドリファレンス」の AWS CloudFormation 「デプロイパラメータ」を参照してください。 AWS CLI 変更セットの詳細については、「 ユーザーガイド」の「変更セットを使用したスタックの更新」を参照してください。 AWS CloudFormation

対応する UI: 設定タブ/詳細/空の変更セットで失敗

disable-rollback

(*DeployCloudFormationStack*/Configuration/disable-rollback)

(オプション)

失敗した場合にスタックデプロイを CodeCatalyst ロールバックするかどうかを指定します。ロールバックは、スタックを既知の最後の安定状態に戻します。デフォルトでは、ロールバックを有効にします。この設定の詳細については、「 コマンドリファレンス」の「 AWS CloudFormation デプロイパラメータ」を参照してください。 AWS CLI

AWS CloudFormation スタックのデプロイアクションがロールバックを処理する方法の詳細については、「」を参照してくださいロールバックの設定。

スタックのロールバックの詳細については、「 ユーザーガイド」の「スタック障害オプションAWS CloudFormation 」を参照してください。

対応する UI: 設定タブ/詳細/ロールバックの無効化

termination-protection

(*DeployCloudFormationStack*/Configuration/termination-protection)

(オプション)

Deploy AWS CloudFormation スタックで、デプロイするスタックに終了保護を追加するかどうかを指定します。削除保護を有効にした状態でスタックを削除しようとすると、削除は失敗し、ステータスを含め、スタックが変更されることはありません。デフォルトでは、終了保護は無効になっています。詳細については、「 AWS CloudFormation ユーザーガイド」の「スタックが削除されないように保護する」を参照してください。

対応する UI: 設定タブ/詳細/終了保護

timeout-in-minutes

(*DeployCloudFormationStack*/Configuration/timeout-in-minutes)

(オプション)

スタック作成オペレーションをタイムアウトし、スタックステータスを に設定するまでに割り当て CloudFormation る時間を分単位で指定しますCREATE_FAILED。が割り当てられた時間内にスタック全体を作成 CloudFormationできない場合、タイムアウトのためにスタックの作成に失敗し、スタックをロールバックします。

デフォルトでは、スタックの作成にタイムアウトはありません。ただし、個々のリソースには、実装するサービスの性質に基づいて、独自のタイムアウトがある可能性があります。例えば、スタック内の個々のリソースがタイムアウトになった場合、スタックの作成に指定されたタイムアウトに到達していない場合でも、スタックの作成もタイムアウトになります。

対応する UI: 設定タブ/詳細/CloudFormationタイムアウト

notification-arns

(*DeployCloudFormationStack*/Configuration/notification-arns)

(オプション)

CodeCatalyst 通知メッセージを送信する Amazon SNS トピックの ARN を指定します。例えば `arn:aws:sns:us-east-1:111222333:MyTopic` です。 AWS CloudFormation スタックのデプロイアクションが実行されると、 CodeCatalyst は と連携して CloudFormation 、スタックの作成または更新プロセス中に発生する AWS CloudFormation イベントごとに 1 つの通知を送信します。(イベントは、スタックの AWS CloudFormation コンソールのイベントタブに表示されます)。最大 5 つのトピックを指定できます。詳細については、「Amazon SNS とは」を参照してください。

対応する UI: 設定タブ/詳細/通知 ARNs

monitor-alarm-arns

(*DeployCloudFormationStack*/Configuration/monitor-alarm-arns)

(オプション)

ロールバックトリガーとして使用する Amazon CloudWatch アラームの Amazon リソースネーム (ARN) を指定します。例えば `arn:aws:cloudwatch::123456789012:alarm/MyAlarm` です。最大 5 つのロールバックトリガーを持つことができます。

> **Note**
>
> CloudWatch アラーム ARN を指定する場合は、アクションが にアクセスできるようにする追加のアクセス許可も設定する必要があります CloudWatch。詳細については、「ロールバックの設定」を参照してください。

対応する UI: 設定タブ/アドバンスト/モニターアラーム ARNs

monitor-timeout-in-minutes

(*DeployCloudFormationStack*/Configuration/monitor-timeout-in-minutes)

(オプション)

が指定されたアラームを CloudFormation モニタリングする 0～180 分までの時間を指定します。モニタリングは、すべてのスタックリソースがデプロイされた後に開始されます。アラームが指定されたモニタリング時間内に発生した場合、デプロイは失敗し、スタックオペレーション全体を CloudFormation ロールバックします。

デフォルト: 0. は、スタックリソースのデプロイ中、後ではなく、アラーム CloudFormation のみをモニタリングします。

対応する UI: 設定タブ/詳細/モニタリング時間

tags

(*DeployCloudFormationStack*/Configuration/tags)

(オプション)

CloudFormation スタックにアタッチするタグを指定します。タグは、コスト配分などの目的でスタックを識別するために使用できる任意のキーと値のペアです。タグとその使用方法の詳細については、「Amazon EC2 ユーザーガイド」の「リソースのタグ付け」を参照してください。でのタグ付けの詳細については CloudFormation、「 AWS CloudFormation ユーザーガイド」のAWS CloudFormation 「スタックオプションの設定」を参照してください。

キーには英数字またはスペースを使用でき、最大 127 文字を使用できます。値には英数字またはスペースを使用でき、最大 255 文字を使用できます。

スタックごとに最大 50 個の一意のタグを追加できます。

対応する UI: 設定タブ/詳細/タグ

ワークフローを使用した AWS Cloud Development Kit (AWS CDK) アプリケーションのデプロイ

このセクションでは、ワークフローを使用して AWS アカウントに AWS CDK アプリケーションをデプロイする方法について説明します。これを実現するには、ワークフローにAWS CDK デプロイアクションを追加する必要があります。AWS CDK デプロイアクションは、 AWS Cloud Development Kit (AWS CDK) アプリケーションを合成して にデプロイします AWS。アプリが にすでに存在する場合 AWS、 アクションは必要に応じてアプリを更新します。

を使用したアプリケーションの記述に関する一般的な情報については AWS CDK、「 AWS Cloud Development Kit (AWS CDK) デベロッパーガイド」の「 とは AWS CDK」を参照してください。

AWS CDK 「デプロイ」アクションを使用するタイミング

を使用してアプリケーションを開発し AWS CDK、自動継続的インテグレーションと配信 (CI/CD) ワークフローの一部として自動的にデプロイする場合は、このアクションを使用します。例えば、誰かが AWS CDK アプリソースに関連するプルリクエストをマージするたびに、 AWS CDK アプリを自動的にデプロイできます。

AWS CDK 「デプロイ」アクションの仕組み

AWS CDK デプロイは次のように機能します。

1. 実行時に、 アクションのバージョン 1.0.12 以前を指定した場合、 アクションは最新の CDK CLI (Tookit とも呼ばれます) AWS CDK を CodeCatalyst ビルドイメージ にダウンロードします。

バージョン 1.0.13 以降を指定した場合、アクションは特定のバージョンの CDK CLI にバンドルされるため、ダウンロードは行われません。

2. アクションは CDK CLI を使用して`cdk deploy`コマンドを実行します。このコマンドは、 AWS CDK アプリケーションを合成して にデプロイします AWS。このコマンドの詳細については、「AWS Cloud Development Kit (AWS CDK) デベロッパーガイド」のAWS CDK 「ツールキット (cdk コマンド)」トピックを参照してください。

「デプロイAWS CDK 」アクションで使用される CDK CLI バージョン

次の表は、AWS CDK デプロイアクションの異なるバージョンでデフォルトで使用されている CDK CLI のバージョンを示しています。

> **Note**
>
> デフォルトを上書きできる場合があります。詳細については、「AWS CDK 「デプロイ」アクション YAML 定義」の「CdkCliVersion」を参照してください。

AWS CDK 「デプロイ」アクションバージョン	AWS CDK CLI バージョン
1.0.0 ~ 1.0.12	最新
1.0.13 以降	2.99.1

アクションをデプロイできるスタックの数

AWS CDK デプロイは 1 つのスタックのみをデプロイできます。 AWS CDK アプリが複数のスタックで構成されている場合は、ネストされたスタックを持つ親スタックを作成し、このアクションを使用して親をデプロイする必要があります。

トピック

- アプリケーションを AWS CDK デプロイするワークフローの例
- AWS CDK 「デプロイ」アクションの追加
- AWS CDK 「デプロイ」アクションによって生成される変数
- AWS CDK 「デプロイ」アクション YAML 定義

アプリケーションを AWS CDK デプロイするワークフローの例

次のワークフローの例には、AWS CDK デプロイアクションとAWS CDK ブートストラップアクションが含まれています。ワークフローは、順番に実行される次の構成要素で構成されます。

> **Note**
>
> 次のワークフロー例は説明を目的としており、追加の設定がないと機能しません。これは、ワークフローがAWS CDK デプロイアクションで設定されている場合のワークフローの例を示すことを目的としています。

- トリガー — このトリガーは、変更をソースリポジトリにプッシュすると、ワークフローの実行を自動的に開始します。このリポジトリには AWS CDK アプリが含まれています。トリガーについての詳細は、「トリガーを使用したワークフローの自動実行の開始」を参照してください。
- AWS CDK ブートストラップアクション (CDKBootstrap) — トリガー時に、アクションはCDKToolkitブートストラップスタックを にデプロイします AWS。CDKToolkit スタックが環境にすでに存在する場合、必要に応じてアップグレードされます。存在しない場合は何も起こらず、アクションは成功としてマークされます。
- AWS CDK デプロイアクション (AWS CDK Deploy) – AWS CDK ブートストラップアクションが完了すると、AWS CDK デプロイアクションは AWS CDK アプリケーションコードを AWS CloudFormation テンプレートに合成し、テンプレートで定義されたスタックを にデプロイします AWS。

```
Name: codecatalyst-cdk-deploy-workflow
SchemaVersion: 1.0

Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  CDKBootstrap:
    Identifier: aws/cdk-bootstrap@v1
    Inputs:
      Sources:
        - WorkflowSource
    Environment:
      Name: codecatalyst-cdk-deploy-environment
```

```
      Connections:
        - Name: codecatalyst-account-connection
          Role: codecatalyst-cdk-bootstrap-role
    Configuration:
      Region: us-west-2

  CDKDeploy:
    Identifier: aws/cdk-deploy@v1
    DependsOn:
      - CDKBootstrap
    Environment:
      Name: codecatalyst-cdk-deploy-environment
      Connections:
        - Name: codecatalyst-account-connection
          Role: codecatalyst-cdk-deploy-role
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      StackName: my-app-stack
      Region: us-west-2
```

AWS CDK 「デプロイ」アクションの追加

次の手順を使用して、ワークフローにAWS CDK デプロイアクションを追加します。

開始する前に

ワークフローにAWS CDK デプロイアクションを追加する前に、次のタスクを完了してください。

1. AWS CDK アプリの準備をします。 AWS CDK v1 または v2 を使用して、 でサポートされている任意のプログラミング言語で AWS CDK アプリケーションを記述できます AWS CDK。 AWS CDK アプリファイルが次の場所で利用可能であることを確認します。
   - CodeCatalyst ソースリポジトリ 、または
   - 別のワークフローアクションによって生成された CodeCatalyst 出力アーティファクト
2. AWS 環境 をブートストラップします。ブートストラップするには、次の操作を行います。
   - 「 デベロッパーガイド」の「ブートストラップ方法」で説明されている方法のいずれかを使用します。 AWS Cloud Development Kit (AWS CDK)
   - AWS CDK ブートストラップアクションを使用します。このアクションは、AWS CDK デプロイと同じワークフロー、または別のワークフローに追加できます。必要なリソースが配置され

るように、AWS CDK デプロイアクションを実行する前に、ブートストラップアクションが少なくとも 1 回実行されていることを確認してください。AWS CDK ブートストラップアクションの詳細については、「」を参照してくださいワークフローを使用した AWS CDK アプリケーションのブートストラップ。

ブートストラップの詳細については、「 AWS Cloud Development Kit (AWS CDK) デベロッパーガイド」の「ブートストラップ」を参照してください。

Visual

ビジュアルエディタを使用してAWS CDK 「デプロイ」アクションを追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. ビジュアル を選択します。
7. 左上で、+ Actions を選択してアクションカタログを開きます。
8. ドロップダウンリストから Amazon CodeCatalystを選択します。
9. AWS CDK デプロイアクションを検索し、次のいずれかを実行します。

   - プラス記号 (+) を選択してワークフロー図にアクションを追加し、設定ペインを開きます。

     または

   - AWS CDK デプロイ を選択します。アクションの詳細ダイアログボックスが表示されます。このダイアログボックスで、次の操作を行います。
     - (オプション) ダウンロード を選択して、アクションのソースコード を表示します。
     - ワークフローに追加 を選択して、ワークフロー図にアクションを追加し、その設定ペインを開きます。
10. 入力タブと設定タブで、必要に応じてフィールドに入力します。各フィールドの説明については、「」を参照してくださいAWS CDK 「デプロイ」アクション YAML 定義。このリファ

レンスでは、YAML エディタとビジュアルエディタの両方に表示される各フィールド (および対応する YAML プロパティ値) に関する詳細情報を提供します。

11. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
12. コミット を選択し、コミットメッセージを入力し、再度コミットを選択します。

> **Note**
>
> AWS CDK デプロイアクションがnpm installエラーで失敗した場合、エラーの修正方法については、「npm install」エラーを修正するにはどうすればよいですか？「」を参照してください。

YAML

YAML エディタを使用してAWS CDK 「デプロイ」アクションを追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. YAML を選択します。
7. 左上で、+ Actions を選択してアクションカタログを開きます。
8. ドロップダウンリストから Amazon CodeCatalystを選択します。
9. AWS CDK デプロイアクションを検索し、次のいずれかを実行します。

   - プラス記号 (+) を選択してワークフロー図にアクションを追加し、設定ペインを開きます。

     または

   - AWS CDK デプロイ を選択します。アクションの詳細ダイアログボックスが表示されます。このダイアログボックスで、次の操作を行います。
     - (オプション) ダウンロード を選択して、アクションのソースコード を表示します。

- ワークフローに追加 を選択して、ワークフロー図にアクションを追加し、その設定ペインを開きます。

10. 必要に応じて YAML コードのプロパティを変更します。使用可能な各プロパティの説明は、「」に記載されていますAWS CDK 「デプロイ」アクション YAML 定義。
11. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
12. コミット を選択し、コミットメッセージを入力し、再度コミットを選択します。

> **Note**
>
> AWS CDK デプロイアクションがnpm installエラーで失敗した場合、エラーの修正方法については、「npm install」エラーを修正するにはどうすればよいですか？「」を参照してください。

AWS CDK 「デプロイ」アクションによって生成される変数

AWS CDK デプロイアクションは、実行時に次の変数を生成して設定します。これらは事前定義された変数 と呼ばれます。

ワークフローでこれらの変数を参照する方法については、「」を参照してください事前定義された変数の使用。

キー	値
スタック ID	ワークフローの実行中に にデプロイされた AWS CDK アプリケーションスタックの Amazon リソースネーム (ARN)。 例: `arn:aws:cloudformation:us-west-2:111122223333:stack/codecatalyst-cdk-app-stack/6aad4380-100a-11ec-a10a-03b8a84d40df`
デプロイプラットフォーム	デプロイプラットフォームの名前。

キー	値
	にハードコードされますAWS:CloudFormation 。
region	ワークフローの実行中に にデプロイ AWS リージョン された のリージョンコード。 例: us-west-2
スキップデプロイメント	の値は、ワークフローの実行中に AWS CDK アプリケーションスタックのデプロイがスキップされたtrueことを示します。前回のデプロイ以降にスタックに変更がない場合、スタックのデプロイはスキップされます。 この変数は、その値が の場合にのみ生成されますtrue。 にハードコードされますtrue。
AWS CloudFormation 変数	AWS CDK デプロイアクションは、前述の変数を生成するだけでなく、CloudFormation出力変数を後続のワークフローアクションで使用するワークフロー変数として公開します。デフォルトでは、 アクションは検出した最初の 4 つ (またはそれ以下の) CloudFormation 変数のみを公開します。どのアクションが公開されているかを判断するには、AWS CDK デプロイアクションを 1 回実行し、実行詳細ページの変数タブを確認します。Variables タブにリストされている変数が目的の変数でない場合は、YAML CfnOutputVariables プロパティを使用して異なる変数を設定できます。詳細については、「」の「 CfnOutputVariablesプロパティの説明」を参照してくださいAWS CDK 「デプロイ」アクション YAML 定義。

AWS CDK 「デプロイ」アクション YAML 定義

以下は、AWS CDK デプロイアクションの YAML 定義です。このアクションの使用方法については、「」を参照してくださいワークフローを使用した AWS Cloud Development Kit (AWS CDK) アプリケーションのデプロイ。

このアクション定義は、より広範なワークフロー定義ファイル内のセクションとして存在します。ファイルの詳細については、「ワークフロー YAML 定義」を参照してください。

> **Note**
>
> 以下の YAML プロパティのほとんどには、対応する UI 要素がビジュアルエディタにあります。UI 要素を検索するには、Ctrl+F を使用します。 要素は、関連付けられた YAML プロパティとともに一覧表示されます。

```
# The workflow definition starts here.
# See ######### for details.

Name: MyWorkflow
SchemaVersion: 1.0
Actions:

# The action definition starts here.
  CDKDeploy_nn:
    Identifier: aws/cdk-deploy@v1
    DependsOn:
      - CDKBootstrap
    Compute:
      Type: EC2 | Lambda
      Fleet: fleet-name
    Timeout: timeout-minutes
    Inputs:
      # Specify a source or an artifact, but not both.
      Sources:
        - source-name-1
      Artifacts:
        - artifact-name
    Outputs:
      Artifacts:
        - Name: cdk_artifact
          Files:
```

```
          - "cdk.out/**/*"
    Environment:
      Name: environment-name
      Connections:
        - Name: account-connection-name
          Role: iam-role-name
    Configuration:
      StackName: my-cdk-stack
      Region: us-west-2
      Tags: '{"key1": "value1", "key2": "value2"}'
      Context: '{"key1": "value1", "key2": "value2"}'
      CdkCliVersion: version
      CdkRootPath: directory-containing-cdk.json-file
      CfnOutputVariables: '["CnfOutputKey1","CfnOutputKey2","CfnOutputKey3"]'
      CloudAssemblyRootPath: path-to-cdk.out
```

CDKDeploy

(必須)

アクションの名前を指定します。すべてのアクション名は、ワークフロー内で一意である必要があります。アクション名は、英数字 (a～z、A～Z、0～9)、ハイフン (-)、アンダースコア (_) に制限されています。スペースは使用できません。引用符を使用してアクション名で特殊文字やスペースを有効にすることはできません。

デフォルト: `CDKDeploy_nn`。

対応する UI: 設定タブ/アクション名

Identifier

(*CDKDeploy*/Identifier)

(必須)

アクションを識別します。バージョンを変更しない限り、このプロパティを変更しないでください。詳細については、「アクションのメジャー、マイナー、またはパッチバージョンの指定」を参照してください。

デフォルト: `aws/cdk-deploy@v1`。

対応する UI: ワークフロー図/CDKDeploy_nn/aws/cdk-deploy@v1 ラベル

DependsOn

(*CDKDeploy*/DependsOn)

(オプション)

AWS CDK デプロイアクションを実行するために正常に実行する必要があるアクションまたはアクショングループを指定します。次のように、 DependsOnプロパティでAWS CDK ブートストラップアクションを指定することをお勧めします。

```
CDKDeploy:
  Identifier: aws/cdk-deploy@v1
  DependsOn:
    - CDKBootstrap
```

> **Note**
>
> ブートストラップは、 AWS CDK アプリケーションをデプロイするための必須の前提条件です。ワークフローにAWS CDK ブートストラップアクションを含めない場合は、デプロイアクションを実行する前に AWS CDK ブートストラップスタックをAWS CDK デプロイする別の方法を見つける必要があります。詳細については、「ワークフローを使用した AWS Cloud Development Kit (AWS CDK) アプリケーションのデプロイ」の AWS CDK 「デプロイ」アクションの追加 を参照してください。

「依存」機能の詳細については、「」を参照してください他のアクションに依存するようにアクションを設定する。

対応する UI: の入力タブ/依存 - オプション

Compute

(*CDKDeploy*/Compute)

(オプション)

ワークフローアクションを実行するために使用されるコンピューティングエンジン。コンピューティングはワークフローレベルまたはアクションレベルで指定できますが、両方を指定することはできません。ワークフローレベルで指定すると、コンピューティング設定はワークフローで定義されたすべてのアクションに適用されます。ワークフローレベルでは、同じインスタンスで複数のアクションを

実行することもできます。詳細については、「アクション間でのコンピューティングの共有」を参照してください。

対応する UI: なし

Type

(*CDKDeploy*/Compute/Type)

(Computeが含まれている場合は必須)

コンピューティングエンジンのタイプ。次のいずれかの値を使用できます。

- EC2 (ビジュアルエディタ) または EC2 (YAML エディタ)

  アクション実行中の柔軟性のために最適化されました。
- Lambda (ビジュアルエディタ) または Lambda (YAML エディタ)

  アクションの起動速度を最適化しました。

コンピューティングタイプの詳細については、「コンピューティングタイプ」を参照してください。

対応する UI: 設定タブ/詳細 - オプション/コンピューティングタイプ

Fleet

(*CDKDeploy*/Compute/Fleet)

(オプション)

ワークフローまたはワークフローアクションを実行するマシンまたはフリートを指定します。オンデマンドフリートでは、アクションが開始されると、ワークフローは必要なリソースをプロビジョニングし、アクションが終了するとマシンは破棄されます。オンデマンドフリートの例: `Linux.x86-64.Large`、`Linux.x86-64.XLarge`。オンデマンドフリートの詳細については、「」を参照してくださいオンデマンドフリートのプロパティ。

プロビジョニングされたフリートでは、ワークフローアクションを実行するように一連の専用マシンを設定します。これらのマシンはアイドル状態のままで、アクションをすぐに処理できます。プロビジョニングされたフリートの詳細については、「」を参照してくださいプロビジョニングされたフリートのプロパティ。

`Fleet` を省略した場合、デフォルトは ですLinux.x86-64.Large。

対応する UI: 設定タブ/アドバンスト - オプション/コンピューティングフリート

Timeout

(*CDKDeploy*/Timeout)

(必須)

がアクション CodeCatalyst を終了するまでにアクションを実行できる時間を分単位で指定します (YAML エディタ)、または時間と分単位で指定します (ビジュアルエディタ)。最小値は 5 分で、最大値は「」で説明されていますワークフローのクォータ。デフォルトのタイムアウトは、最大タイムアウトと同じです。

対応する UI: 設定タブ/タイムアウト - オプション

Inputs

(*CDKDeploy*/Inputs)

(オプション)

`Inputs` セクションでは、ワークフローの実行中に がCDKDeploy必要とするデータを定義します。

> **Note**
>
> AWS CDK デプロイアクションごとに許可される入力 (ソースまたはアーティファクト) は 1 つだけです。

対応する UI: Inputs タブ

Sources

(*CDKDeploy*/Inputs/Sources)

(デプロイする AWS CDK アプリがソースリポジトリに保存されている場合に必須)

AWS CDK アプリケーションがソースリポジトリに保存されている場合は、そのソースリポジトリのラベルを指定します。AWS CDK デプロイアクションは、デプロイプロセスを開始する前に、

このリポジトリ内のアプリケーションを合成します。現在、サポートされているラベルは のみです`WorkflowSource`。

AWS CDK アプリがソースリポジトリに含まれていない場合は、別のアクションによって生成されたアーティファクトに存在する必要があります。

sources の詳細については、「ワークフローをソースリポジトリに接続する」を参照してください。

対応する UI: 入力タブ/ソース - オプション

Artifacts - input

(*CDKDeploy*/Inputs/Artifacts)

(デプロイする AWS CDK アプリケーションが前のアクションの出力アーティファクトに保存されている場合に必要です)

前のアクションで生成されたアーティファクトに AWS CDK アプリが含まれている場合は、ここでそのアーティファクトを指定します。AWS CDK デプロイアクションは、デプロイプロセスを開始する前に、指定されたアーティファクト内のアプリケーションをテンプレートに CloudFormation合成します。 AWS CDK アプリがアーティファクトに含まれていない場合は、ソースリポジトリに存在する必要があります。

アーティファクトの例などの詳細については、「」を参照してくださいアーティファクトを使用したワークフロー内のアクション間のデータの共有。

対応する UI: 入力タブ/アーティファクト - オプション

Outputs

(*CDKDeploy*/Outputs)

(オプション)

ワークフローの実行中に アクションによって出力されるデータを定義します。

対応する UI: 出力タブ

Artifacts - output

(*CDKDeploy*/Outputs/Artifacts

(オプション)

アクションによって生成されたアーティファクトを指定します。これらのアーティファクトは、他のアクションで入力として参照できます。

アーティファクトの例などの詳細については、「」を参照してくださいアーティファクトを使用したワークフロー内のアクション間のデータの共有。

対応する UI: 出力タブ/アーティファクト

Name

(*CDKDeploy*/Outputs/Artifacts/Name)

(Artifacts - outputが含まれている場合は必須)

実行時にAWS CDK デプロイアクションによって合成される AWS CloudFormation テンプレートを含むアーティファクトの名前を指定します。デフォルト値は、cdk_artifactです。アーティファクトを指定しない場合、アクションはテンプレートを合成しますが、アーティファクトには保存されません。テストまたはトラブルシューティングの目的で、合成されたテンプレートをアーティファクトに保存して、そのレコードを保存することを検討してください。

対応する UI: タブ/アーティファクト/アーティファクト名の追加/ビルドの出力

Files

(*CDKDeploy*/Outputs/Artifacts/Files)

(Artifacts - outputが含まれている場合は必須)

アーティファクトに含めるファイルを指定します。 AWS CDK アプリの合成 AWS CloudFormation テンプレートを含める"cdk.out/**/*"には、 を指定する必要があります。

> **Note**
>
> cdk.out は、合成されたファイルが保存されるデフォルトのディレクトリです。cdk.json ファイルcdk.outで 以外の出力ディレクトリを指定した場合は、 ではなく、ここでそのディレクトリを指定しますcdk.out。

対応する UI: タブ/アーティファクトを出力/ビルドによって生成されたアーティファクト/ファイルを追加

Environment

(*CDKDeploy*/Environment)

(必須)

アクションで使用する CodeCatalyst 環境を指定します。

環境の詳細については、環境を使用して AWS アカウント および VPCsにデプロイする CodeCatalyst 「」および「」を参照してください環境を作成する。

対応する UI: 設定タブ/'環境/アカウント/ロール'/環境

Name

(*CDKDeploy*/Environment/Name)

(Environmentが含まれている場合は必須)

アクションに関連付ける既存の環境の名前を指定します。

対応する UI: 設定タブ/'環境/アカウント/ロール'/環境

Connections

(*CDKDeploy*/Environment/Connections)

(Environmentが含まれている場合は必須)

アクションに関連付けるアカウント接続を指定します。で最大 1 つのアカウント接続を指定できますEnvironment。

アカウント接続の詳細については、「」を参照してください接続された AWS リソースへのアクセスを許可する AWS アカウント。アカウント接続を環境に関連付ける方法については、「」を参照してください環境を作成する。

対応する UI: 設定タブ/'環境/アカウント/ロール'/AWS アカウント接続

Name

(*CDKDeploy*/Environment/Connections/Name)

(必須)

アカウント接続の名前を指定します。

対応する UI: 設定タブ/'環境/アカウント/ロール'/AWS アカウント接続

Role

(*CDKDeploy*/Environment/Connections/Role)

(必須)

AWS CDK デプロイアクションが AWS CDK アプリケーションスタックにアクセスして AWS デプロイするために使用する IAM ロールの名前を指定します。このロールに以下が含まれていることを確認してください。

- 次のアクセス許可ポリシー：

> **Warning**
>
> アクセス許可は、次のポリシーに示すものに制限します。より広範なアクセス許可を持つロールを使用すると、セキュリティ上のリスクが生じる可能性があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "cloudformation:DescribeStackEvents",
                "cloudformation:DescribeChangeSet",
                "cloudformation:DescribeStacks",
                "cloudformation:ListStackResources"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::aws-account:role/cdk-*"
        }
    ]
}
```

- 次のカスタム信頼ポリシー：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

このロールがアカウント接続に追加されていることを確認します。アカウント接続への IAM ロールの追加の詳細については、「」を参照してくださいアカウント接続への IAM ロールの追加。

> **Note**
>
> 必要に応じて、ここでCodeCatalystWorkflowDevelopmentRole-*spaceName*ロールの名前を指定できます。このロールの詳細については、「アカウントとスペースのCodeCatalystWorkflowDevelopmentRole-***spaceName***ロールの作成」を参照してください。CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールには、セキュリティリスクをもたらす可能性のある非常に広範なアクセス許可があることを理解します。このロールは、セキュリティが懸念されないチュートリアルとシナリオでのみ使用することをお勧めします。

対応する UI: 設定タブ/'環境/アカウント/ロール'/ロール

Configuration

(*CDKDeploy*/Configuration)

(必須)

アクションの設定プロパティを定義できるセクション。

対応する UI: 設定タブ

StackName

(*CDKDeploy*/Configuration/StackName)

(必須)

AWS CDK アプリの bin ディレクトリのエントリポイントファイルに表示される AWS CDK アプリスタックの名前。次の例は、 TypeScriptエントリポイントファイルの内容を示し、スタック名が*####* で強調表示されています。エントリポイントファイルが別の言語にある場合、そのファイルは似ています。

```
import * as cdk from 'aws-cdk-lib';
import { CdkWorksopTypescriptStack } from '../lib/cdk_workshop_typescript-stack';

const app = new cdk.App();
new CdkWorkshopTypescriptStack(app, 'CdkWorkshopTypescriptStack');
```

指定できるスタックは 1 つだけです。

> **Tip**
> 複数のスタックがある場合は、ネストされたスタックを持つ親スタックを作成できます。その後、このアクションで親スタックを指定して、すべてのスタックをデプロイできます。

対応する UI: 設定タブ/スタック名

Region

(*CDKDeploy*/Configuration/Region)

(必須)

AWS CDK アプリケーションスタックをデプロイ AWS リージョン する を指定します。リージョンコードの一覧については、「リージョンエンドポイント」を参照してください。

対応する UI: 設定タブ/リージョン

Tags

(*CDKDeploy*/Configuration/Tags)

(オプション)

AWS CDK アプリケーションスタックの AWS リソースに適用するタグを指定します。タグは、スタック自体とスタック内の個々のリソースに適用されます。タグ付けの詳細については、「 AWS Cloud Development Kit (AWS CDK) デベロッパーガイド」の「タグ付け」を参照してください。

対応する UI: 設定タブ/詳細 - オプション/タグ

Context

(*CDKDeploy*/Configuration/Context)

(オプション)

AWS CDK アプリケーションスタックに関連付けるコンテキストをキーと値のペアの形式で指定します。コンテキストの詳細については、「 AWS Cloud Development Kit (AWS CDK) デベロッパーガイド」の「ランタイムコンテキスト」を参照してください。

対応する UI: 設定タブ/詳細 - オプション/コンテキスト

CdkCliVersion

(*CDKDeploy*/Configuration/CdkCliVersion)

(オプション)

このプロパティは、AWS CDK デプロイアクションのバージョン 1.0.13 以降、およびAWS CDK ブートストラップアクションのバージョン 1.0.8 以降で使用できます。

次のいずれかを指定します。

- このアクションで使用する AWS Cloud Development Kit (AWS CDK) コマンドラインインターフェイス (CLI) (ツールキットとも呼ばれます AWS CDK) のフルバージョン。例えば、`2.102.1` などです。アプリケーションの構築とデプロイ時に一貫性と安定性を確保するために、フルバージョンを指定することを検討してください。

  または

- `latest`。 CDK CLI の最新の機能と修正を活用する`latest`には、 を指定することを検討してください。

アクションは、指定されたバージョンの AWS CDK CLI (または最新バージョン) を CodeCatalyst ビルドイメージ にダウンロードし、このバージョンを使用して CDK アプリケーションのデプロイまたは環境のブートストラップ AWS に必要なコマンドを実行します。

使用できるサポートされている CDK CLI バージョンのリストについては、AWS CDK 「バージョン」を参照してください。

このプロパティを省略すると、 アクションは、次のいずれかのトピックで説明されているデフォルトの AWS CDK CLI バージョンを使用します。

- 「デプロイAWS CDK 」アクションで使用される CDK CLI バージョン
- AWS CDK 「ブートストラップ」アクションで使用される CDK CLI バージョン

対応する UI: 設定タブ/AWS CDK CLI バージョン

CdkRootPath

(*CDKDeploy*/Configuration/CdkRootPath)

(オプション)

AWS CDK プロジェクトの `cdk.json` ファイルを含むディレクトリへのパス。AWS CDK デプロイアクションはこのフォルダから実行され、アクションによって作成された出力はこのディレクトリに追加されます。指定しない場合、AWS CDK デプロイアクションは`cdk.json`ファイルが AWS CDK プロジェクトのルートにあることを前提としています。

対応する UI: cdk.json が存在する設定タブ/ディレクトリ

CfnOutputVariables

(*CDKDeploy*/Configuration/CfnOutputVariables)

(オプション)

ワークフロー出力変数として公開する AWS CDK アプリケーションコード内の`CfnOutput`コンストラクトを指定します。その後、ワークフロー内の後続のアクションでワークフロー出力変数を参照できます。の変数の詳細については、 CodeCatalyst「」を参照してくださいワークフローでの変数の設定と使用。

例えば、 AWS CDK アプリケーションコードは次のようになります。

```
import { Duration, Stack, StackProps, CfnOutput, RemovalPolicy} from 'aws-cdk-lib';
import * as dynamodb from 'aws-cdk-lib/aws-dynamodb';
import * as s3 from 'aws-cdk-lib/aws-s3';
import { Construct } from 'constructs';
import * as cdk from 'aws-cdk-lib';
export class HelloCdkStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);
    const bucket = new s3.Bucket(this, 'my-bucket', {
      removalPolicy: RemovalPolicy.DESTROY,
    });
    new CfnOutput(this, 'bucketName', {
      value: bucket.bucketName,
      description: 'The name of the s3 bucket',
      exportName: 'myBucket',
    });
    const table = new dynamodb.Table(this, 'todos-table', {
      partitionKey: {name: 'todoId', type: dynamodb.AttributeType.NUMBER},
      billingMode: dynamodb.BillingMode.PAY_PER_REQUEST,
      removalPolicy: RemovalPolicy.DESTROY,
    })
    new CfnOutput(this, 'tableName', {
      value: table.tableName,
      description: 'The name of the dynamodb table',
      exportName: 'myDynamoDbTable',
    });
    ...
  }
}
```

プロパティ`CfnOutputVariables`は次のようになります。

```
Configuration:
  ...
  CfnOutputVariables: '["bucketName","tableName"]'
```

アクションは、次のワークフロー出力変数を生成します。

キー	値
bucketName	`bucket.bucketName`

キー	値
tableName	`table.tableName`

その後、後続のアクションで `bucketName`および `tableName`変数を参照できます。後続のアクションでワークフロー出力変数を参照する方法については、「」を参照してください事前定義された変数の参照。

`CfnOutputVariables` プロパティに`CfnOutput`コンストラクトを指定しない場合、 アクションはワークフロー出力変数として見つかった最初の 4 つ (またはそれ以下の) CloudFormation の出力変数を公開します。詳細については、「AWS CDK 「デプロイ」アクションによって生成される変数」を参照してください。

> **Tip**
>
> アクションが生成するすべての CloudFormation 出力変数のリストを取得するには、AWS CDK デプロイアクションを含むワークフローを 1 回実行し、アクションのログタブを確認します。ログには、 AWS CDK アプリに関連付けられているすべての CloudFormation 出力変数のリストが含まれます。すべての CloudFormation 変数が何であるかがわかったら、`CfnOutputVariables`プロパティを使用してワークフロー出力変数に変換する変数を指定できます。

AWS CloudFormation 出力変数の詳細については、 AWS Cloud Development Kit (AWS CDK) API リファレンスの クラス (`CfnOutput`コンストラクト) で利用可能な コンストラクトのドキュメントを参照してください。 CfnOutput

対応する UI: 設定タブ/AWS CloudFormation 出力変数

CloudAssemblyRootPath

(*`CDKDeploy`*/Configuration/CloudAssemblyRootPath)

(オプション)

AWS CDK アプリのスタックを (`cdk synth`オペレーションを使用して) クラウドアセンブリに既に合成している場合は、クラウドアセンブリディレクトリのルートパス () を指定します`cdk.out`。指定されたクラウドアセンブリディレクトリにある AWS CloudFormation テンプレートは、 `cdk deploy --app` コマンド AWS アカウント を使用して にデプロイアクションによってAWS CDK デ

プロイされます。--app オプションが存在する場合、cdk synthオペレーションは実行されません。

クラウドアセンブリディレクトリを指定しない場合、AWS CDK デプロイアクションは --appオプションを指定せずに cdk deploy コマンドを実行します。--app オプションを指定しない場合、cdk deployオペレーションは (cdk synth) を合成し、 に AWS CDK アプリケーションをデプロイします AWS アカウント。

AWS CDK 「deploy」アクションが実行時に合成できるときに、既存の合成されたクラウドアセンブリを指定するのはなぜですか？

合成された既存のクラウドアセンブリを次のように指定できます。

- 「deploy」アクションを実行するたびに、まったく同じリソースセットがAWS CDK デプロイされていることを確認します。

  クラウドアセンブリを指定しない場合、AWS CDK デプロイアクションは実行時期に応じて異なるファイルを合成してデプロイできます。例えば、AWS CDK デプロイアクションは、テスト段階では 1 つの依存関係のセット、本番稼働段階では別の依存関係のセット (ステージ間で依存関係が変更された場合) を使用してクラウドアセンブリを合成する場合があります。テスト対象とデプロイ対象との正確なパリティを保証するには、一度合成してから、クラウドアセンブリへのパスディレクトリフィールド (ビジュアルエディタ) またはCloudAssemblyRootPathプロパティ (YAML エディタ) を使用して、既に合成されたクラウドアセンブリを指定することをお勧めします。
- 非標準のパッケージマネージャーとツールを AWS CDK アプリで使用する

  synth オペレーション中、AWS CDK デプロイアクションは npm や pip などの標準ツールを使用してアプリを実行しようとします。アクションがこれらのツールを使用してアプリを正常に実行できない場合、合成は行われず、アクションは失敗します。この問題を回避するには、アプリの cdk.json ファイルで AWS CDK アプリを正常に実行するために必要な正確なコマンドを指定し、AWS CDK デプロイアクションを含まないメソッドを使用してアプリを合成します。クラウドアセンブリが生成されたら、AWS CDK デプロイアクションのクラウドアセンブリへのパスディレクトリフィールド (ビジュアルエディタ) またはCloudAssemblyRootPathプロパティ (YAML エディタ) で指定できます。

AWS CDK アプリをインストールして実行するためのコマンドを含めるように cdk.json ファイルを設定する方法については、「アプリコマンドの指定」を参照してください。

コマンドcdk deployと cdk synth コマンド、および --appオプションの詳細については、「デベロッパーガイド」の「スタックのデプロイ」、「スタックの合成」、「合成のスキップ」を参照

してください。 https://docs.aws.amazon.com/cdk/v2/guide/cli.html#cli-deploy-nosynth AWS Cloud Development Kit (AWS CDK)

クラウドアセンブリの詳細については、AWS Cloud Development Kit (AWS CDK) 「 API リファレンス」の「Cloud Assembly」を参照してください。

対応する UI: 設定タブ/クラウドアセンブリディレクトリへのパス

ワークフローを使用した AWS CDK アプリケーションのブートストラップ

このセクションでは、 CodeCatalyst ワークフローを使用して AWS CDK アプリケーションをブートストラップする方法について説明します。これを実現するには、ワークフローにAWS CDK ブートストラップアクションを追加する必要があります。AWS CDK ブートストラップアクションは、最新のテンプレート を使用して、環境にブートストラップスタックをプロビジョニングします AWS 。ブートストラップスタックが既に存在する場合、 アクションは必要に応じてそれを更新します。にブートストラップスタックが存在することは、 AWS CDK アプリケーションをデプロイするための前提条件 AWS です。

ブートストラップの詳細については、「 AWS Cloud Development Kit (AWS CDK) デベロッパーガイド」の「ブートストラップ」を参照してください。

AWS CDK 「ブートストラップ」アクションを使用するタイミング

AWS CDK アプリケーションをデプロイするワークフローがあり、ブートストラップスタックを同時にデプロイ (および必要に応じて更新) する場合は、このアクションを使用します。この場合、 AWS CDK アプリをデプロイするワークフローと同じワークフローにAWS CDK ブートストラップアクションを追加します。

次のいずれかに当てはまる場合は、このアクションを使用しないでください。

- 別のメカニズムを使用してブートストラップスタックを既にデプロイしており、そのままにしておく (更新なし) 必要があります。
- ブートストラップアクションではサポートされていないカスタムブートストラップテンプレート を使用します。 AWS CDK

AWS CDK 「ブートストラップ」アクションの仕組み

AWS CDK ブートストラップは次のように機能します。

1. 実行時に、 アクションのバージョン 1.0.7 以前を指定した場合、 アクションは最新の CDK CLI (Tookit とも呼ばれます) AWS CDK を CodeCatalyst ビルドイメージ にダウンロードします。

   バージョン 1.0.8 以降を指定した場合、アクションは特定のバージョンの CDK CLI にバンドルされるため、ダウンロードは行われません。

2. アクションは CDK CLI を使用して`cdk bootstrap`コマンドを実行します。このコマンドは、AWS Cloud Development Kit (AWS CDK) デベロッパーガイドの「ブートストラップ」トピックで説明されているブートストラップタスクを実行します。

AWS CDK 「ブートストラップ」アクションで使用される CDK CLI バージョン

次の表は、AWS CDK ブートストラップアクションのさまざまなバージョンでデフォルトで使用されている CDK CLI のバージョンを示しています。

> **Note**
>
> デフォルトを上書きできる場合があります。詳細については、「AWS CDK 「ブートストラップ」アクション YAML 定義」の「CdkCliVersion」を参照してください。

AWS CDK 「ブートストラップ」アクション バージョン	AWS CDK CLI バージョン
1.0.0 ~ 1.0.7	最新
1.0.8 以降	2.99.1

トピック

- AWS CDK アプリケーションをブートストラップするワークフローの例
- AWS CDK 「ブートストラップ」アクションの追加
- AWS CDK 「ブートストラップ」アクションによって生成される変数
- AWS CDK 「ブートストラップ」アクション YAML 定義

AWS CDK アプリケーションをブートストラップするワークフローの例

ブートストラップアクションを含むワークフローワークフローを使用した AWS Cloud Development Kit (AWS CDK) アプリケーションのデプロイについては、アプリケーションを AWS CDK デプロイするワークフローの例「」の「」を参照してください。 AWS CDK

AWS CDK 「ブートストラップ」アクションの追加

次の手順を使用して、ワークフローにAWS CDK ブートストラップアクションを追加します。

開始する前に

AWS CDK ブートストラップアクションを使用する前に、 AWS CDK アプリの準備が完了していることを確認してください。ブートストラップアクションは、ブートストラップ前に AWS CDK アプリケーションを合成します。アプリは、 でサポートされている任意のプログラミング言語で記述できます AWS CDK。

AWS CDK アプリファイルが次の場所で利用可能であることを確認します。

- CodeCatalyst ソースリポジトリ 、または
- 別のワークフローアクションによって生成された CodeCatalyst 出力アーティファクト

Visual

ビジュアルエディタを使用してAWS CDK 「ブートストラップ」アクションを追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. ビジュアル を選択します。
7. 左上で、+ Actions を選択してアクションカタログを開きます。
8. ドロップダウンリストから Amazon CodeCatalystを選択します。
9. AWS CDK ブートストラップアクションを検索し、次のいずれかを実行します。

- プラス記号 (+) を選択してワークフロー図にアクションを追加し、その設定ペインを開きます。

  または

- AWS CDK ブートストラップ を選択します。アクションの詳細ダイアログボックスが表示されます。このダイアログボックスで、次の操作を行います。
  - (オプション) ソースを表示 を選択して、アクションのソースコード を表示します。
  - ワークフローに追加 を選択して、ワークフロー図にアクションを追加し、その設定ペインを開きます。

10. Inputs 、Configuration 、および Outputs タブで、必要に応じてフィールドに入力します。各フィールドの説明については、「」を参照してくださいAWS CDK 「ブートストラップ」アクション YAML 定義。このリファレンスでは、YAML エディタとビジュアルエディタの両方に表示される各フィールド (および対応する YAML プロパティ値) に関する詳細情報を提供します。
11. (オプション) Validate を選択して、コミットする前にワークフローの YAML コードを検証します。
12. コミット を選択し、コミットメッセージを入力し、再度コミットを選択します。

> **Note**
>
> AWS CDK ブートストラップアクションがエラーで失敗した場合、エラーの修正方法については、「npm install」エラーを修正するにはどうすればよいですか？「」を参照してください。 `npm install`

YAML

YAML エディタを使用してAWS CDK 「ブートストラップ」アクションを追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。

5. [編集] を選択します。
6. YAML を選択します。
7. 左上で、+ Actions を選択してアクションカタログを開きます。
8. ドロップダウンリストから Amazon CodeCatalystを選択します。
9. AWS CDK ブートストラップアクションを検索し、 + を選択してワークフロー図に追加し、設定ペインを開きます。
10. 必要に応じて YAML コードのプロパティを変更します。使用可能な各プロパティの説明は、「」に記載されていますAWS CDK「ブートストラップ」アクション YAML 定義。
11. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
12. コミット を選択し、コミットメッセージを入力し、再度コミットを選択します。

> **Note**
>
> AWS CDK ブートストラップアクションがエラーで失敗した場合、エラーの修正方法については、「npm install」エラーを修正するにはどうすればよいですか？「」を参照してください。 `npm install`

AWS CDK「ブートストラップ」アクションによって生成される変数

AWS CDK ブートストラップアクションは、実行時に次の変数を生成して設定します。これらは事前定義された変数 と呼ばれます。

ワークフローでこれらの変数を参照する方法については、「」を参照してください事前定義された変数の使用。

キー	値
デプロイプラットフォーム	デプロイプラットフォームの名前。 にハードコードされていますAWS:CloudFormation 。
region	ワークフローの実行中に AWS CDK ブートストラップスタック AWS リージョン がデプロイされた のリージョンコード。

キー	値
	例: `us-west-2`
スタック ID	デプロイされた AWS CDK ブートストラップスタックの Amazon リソースネーム (ARN)。 例: `arn:aws:cloudformation:us-west-2:111122223333:stack/codecatalyst-cdk-bootstrap-stack/6aad4380-100a-11ec-a10a-03b8a84d40df`
スキップデプロイメント	の値は、ワークフローの実行中に AWS CDK ブートストラップスタックのデプロイがスキップされた`true`ことを示します。前回のデプロイ以降にスタックに変更がない場合、スタックのデプロイはスキップされます。 この変数は、その値が の場合にのみ生成されます`true`。 にハードコードされます`true`。

AWS CDK 「ブートストラップ」アクション YAML 定義

以下は、AWS CDK ブートストラップアクションの YAML 定義です。このアクションの使用方法については、「」を参照してくださいワークフローを使用した AWS CDK アプリケーションのブートストラップ。

このアクション定義は、より広範なワークフロー定義ファイル内のセクションとして存在します。ファイルの詳細については、「ワークフロー YAML 定義」を参照してください。

> **Note**
>
> 以下の YAML プロパティのほとんどには、対応する UI 要素がビジュアルエディタにあります。UI 要素を検索するには、Ctrl+F を使用します。 要素は、関連付けられた YAML プロパティとともに一覧表示されます。

```
# The workflow definition starts here.
# See ######### for details.

Name: MyWorkflow
SchemaVersion: 1.0
Actions:

# The action definition starts here.
  CDKBootstrapAction_nn:
    Identifier: aws/cdk-bootstrap@v1
    DependsOn:
      - action-name
    Compute:
      Type: EC2 | Lambda
      Fleet: fleet-name
    Timeout: timeout-minutes
    Inputs:
      # Specify a source or an artifact, but not both.
      Sources:
        - source-name-1
      Artifacts:
        - artifact-name
    Outputs:
      Artifacts:
        - Name: cdk_bootstrap_artifacts
          Files:
            - "cdk.out/**/*"
    Environment:
      Name: environment-name
      Connections:
        - Name: account-connection-name
          Role: iam-role-name
    Configuration:
      Region: us-west-2
      CdkCliVersion: version
```

CDKBootstrapAction

(必須)

アクションの名前を指定します。すべてのアクション名は、ワークフロー内で一意である必要があります。アクション名は、英数字 (a ~ z、A ~ Z、0 ~ 9)、ハイフン (-)、アンダースコア (_) に制限され

ています。スペースは使用できません。引用符を使用してアクション名で特殊文字やスペースを有効にすることはできません。

デフォルト: `CDKBootstrapAction_nn`。

対応する UI: 設定タブ/アクションの表示名

Identifier

(*CDKBootstrapAction*/Identifier)

(必須)

アクションを識別します。バージョンを変更しない限り、このプロパティを変更しないでください。詳細については、「アクションのメジャー、マイナー、またはパッチバージョンの指定」を参照してください。

デフォルト: `aws/cdk-bootstrap@v1`。

対応する UI: ワークフロー図/CDKBootstrapAction_nn/aws/cdk-bootstrap@v1 ラベル

DependsOn

(*CDKBootstrapAction*/DependsOn)

(オプション)

このアクションを実行するために正常に実行する必要があるアクション、アクショングループ、またはゲートを指定します。

「依存」機能の詳細については、「」を参照してください他のアクションに依存するようにアクションを設定する。

対応する UI: の入力タブ/依存 - オプション

Compute

(*CDKBootstrapAction*/Compute)

(オプション)

ワークフローアクションを実行するために使用されるコンピューティングエンジン。コンピューティングはワークフローレベルまたはアクションレベルで指定できますが、両方を指定することはできません。ワークフローレベルで指定すると、コンピューティング設定はワークフローで定義されたすべてのアクションに適用されます。ワークフローレベルでは、同じインスタンスで複数のアクションを

実行することもできます。詳細については、「アクション間でのコンピューティングの共有」を参照してください。

対応する UI: なし

Type

(*CDKBootstrapAction*/Compute/Type)

(Computeが含まれている場合は必須)

コンピューティングエンジンのタイプ。次のいずれかの値を使用できます。

- EC2 (ビジュアルエディタ) または EC2 (YAML エディタ)

  アクション実行中の柔軟性のために最適化されました。
- Lambda (ビジュアルエディタ) または Lambda (YAML エディタ)

  アクションの起動速度を最適化しました。

コンピューティングタイプの詳細については、「コンピューティングタイプ」を参照してください。

対応する UI: 設定タブ/詳細 - オプション/コンピューティングタイプ

Fleet

(*CDKBootstrapAction*/Compute/Fleet)

(オプション)

ワークフローまたはワークフローアクションを実行するマシンまたはフリートを指定します。オンデマンドフリートでは、アクションが開始されると、ワークフローは必要なリソースをプロビジョニングし、アクションが終了するとマシンは破棄されます。オンデマンドフリートの例: `Linux.x86-64.Large`、`Linux.x86-64.XLarge`。オンデマンドフリートの詳細については、「」を参照してくださいオンデマンドフリートのプロパティ。

プロビジョニングされたフリートでは、ワークフローアクションを実行するように一連の専用マシンを設定します。これらのマシンはアイドル状態のままで、アクションをすぐに処理できます。プロビジョニングされたフリートの詳細については、「」を参照してくださいプロビジョニングされたフリートのプロパティ。

`Fleet` を省略した場合、デフォルトは ですLinux.x86-64.Large。

対応する UI: 設定タブ/アドバンスト - オプション/コンピューティングフリート

Timeout

(*CDKBootstrapAction*/Timeout)

(必須)

がアクション CodeCatalyst を終了するまでにアクションを実行できる時間を分単位で指定します (YAML エディタ)、または時間と分単位で指定します (ビジュアルエディタ)。最小値は 5 分で、最大値は「」で説明されていますワークフローのクォータ。デフォルトのタイムアウトは、最大タイムアウトと同じです。

対応する UI: 設定タブ/タイムアウト - オプション

Inputs

(*CDKBootstrapAction*/Inputs)

(オプション)

`Inputs` セクションでは、ワークフローの実行中にAWS CDK ブートストラップアクションが必要とするデータを定義します。

対応する UI: Inputs タブ

> **Note**
>
> AWS CDK ブートストラップアクションごとに許可される入力 (ソースまたはアーティファクト) は 1 つだけです。

Sources

(*CDKBootstrapAction*/Inputs/Sources)

(AWS CDK アプリがソースリポジトリに保存されている場合に必須)

AWS CDK アプリケーションがソースリポジトリに保存されている場合は、そのソースリポジトリのラベルを指定します。AWS CDK ブートストラップアクションは、ブートストラッププロセスを開始する前に、このリポジトリ内のアプリケーションを合成します。現在、サポートされているリポジトリラベルは のみです`WorkflowSource`。

AWS CDK アプリがソースリポジトリに含まれていない場合は、別のアクションによって生成されたアーティファクトに存在する必要があります。

sources の詳細については、「ワークフローをソースリポジトリに接続する」を参照してください。

対応する UI: 入力タブ/ソース - オプション

Artifacts - input

(*CDKBootstrapAction*/Inputs/Artifacts)

(AWS CDK アプリが前のアクションの出力アーティファクトに保存されている場合に必要です)

前のアクションで生成されたアーティファクトに AWS CDK アプリが含まれている場合は、ここでそのアーティファクトを指定します。AWS CDK ブートストラップアクションは、ブートストラッププロセスを開始する前に、指定されたアーティファクト内のアプリケーションを CloudFormation テンプレートに合成します。 AWS CDK アプリがアーティファクトに含まれていない場合は、ソースリポジトリに存在する必要があります。

アーティファクトの例などの詳細については、「」を参照してくださいアーティファクトを使用したワークフロー内のアクション間のデータの共有。

対応する UI: 入力タブ/アーティファクト - オプション

Outputs

(*CDKBootstrapAction*/Outputs)

(オプション)

ワークフローの実行中に アクションによって出力されるデータを定義します。

対応する UI: 出力タブ

Artifacts - output

(*CDKBootstrapAction*/Outputs/Artifacts)

(オプション)

アクションによって生成されたアーティファクトを指定します。これらのアーティファクトは、他のアクションで入力として参照できます。

アーティファクトの例などの詳細については、「」を参照してくださいアーティファクトを使用したワークフロー内のアクション間のデータの共有。

対応する UI: 出力タブ/アーティファクト

Name

(*CDKBootstrapAction*/Outputs/Artifacts/Name)

(Artifacts - outputが含まれている場合は必須)

実行時にAWS CDK ブートストラップアクションによって合成される AWS CloudFormation テンプレートを含むアーティファクトの名前を指定します。デフォルト値は、cdk_bootstrap_artifactsです。アーティファクトを指定しない場合、アクションはテンプレートを合成しますが、アーティファクトには保存されません。テストまたはトラブルシューティングの目的で、合成されたテンプレートをアーティファクトに保存して、そのレコードを保存することを検討してください。

対応する UI: 出力タブ/アーティファクト/アーティファクト名の追加/アーティファクト名の構築

Files

(*CDKBootstrapAction*/Outputs/Artifacts/Files)

(Artifacts - outputが含まれている場合は必須)

アーティファクトに含めるファイルを指定します。 AWS CDK アプリの合成 AWS CloudFormation テンプレートを含める"cdk.out/**/*"には、 を指定する必要があります。

> **Note**
>
> cdk.out は、合成されたファイルが保存されるデフォルトのディレクトリです。cdk.json ファイルcdk.outで 以外の出力ディレクトリを指定した場合は、 ではなく、ここでそのディレクトリを指定しますcdk.out。

対応する UI: タブ/アーティファクトを出力/ビルドによって生成されたアーティファクト/ファイルを追加

Environment

(*CDKBootstrapAction*/Environment)

(必須)

アクションで使用する CodeCatalyst 環境を指定します。

環境の詳細については、環境を使用して AWS アカウント および VPCsにデプロイする CodeCatalyst「」および「」を参照してください環境を作成する。

対応する UI: 設定タブ/'環境/アカウント/ロール'/環境

Name

(*CDKBootstrapAction*/Environment/Name)

(Environmentが含まれている場合は必須)

アクションに関連付ける既存の環境の名前を指定します。

対応する UI: 設定タブ/'環境/アカウント/ロール'/環境

Connections

(*CDKBootstrapAction*/Environment/Connections)

(Environmentが含まれている場合は必須)

アクションに関連付けるアカウント接続を指定します。で最大 1 つのアカウント接続を指定できますEnvironment。

アカウント接続の詳細については、「」を参照してください接続された AWS リソースへのアクセスを許可する AWS アカウント。アカウント接続を環境に関連付ける方法については、「」を参照してください環境を作成する。

対応する UI: 設定タブ/'環境/アカウント/ロール'/AWS アカウント接続

Name

(*CDKBootstrapAction*/Environment/Connections/Name)

(必須)

アカウント接続の名前を指定します。

対応する UI: 設定タブ/'環境/アカウント/ロール'/AWS アカウント接続

Role

(*CDKBootstrapAction*/Environment/Connections/Role)

(必須)

AWS CDK ブートストラップアクションがブートストラップスタックにアクセスして AWS 追加するために使用する IAM ロールの名前を指定します。このロールに次のポリシーが含まれていることを確認します。

> **Note**
>
> 次のアクセス許可ポリシーに示されているアクセス許可は、ブートストラップを実行するために cdk bootstrap コマンドで必要なアクセス許可です。これらのアクセス許可は、 がブートストラップコマンド AWS CDK を変更すると変更される場合があります。

> **Warning**
>
> このロールはAWS CDK ブートストラップアクションでのみ使用します。これは非常に許容されており、他のアクションで使用するとセキュリティリスクが生じる可能性があります。

- 次のアクセス許可ポリシー：

  > **Warning**
  >
  > アクセス許可は、次のポリシーに示すものに制限します。より広範なアクセス許可を持つロールを使用すると、セキュリティ上のリスクが生じる可能性があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "ssm:GetParameterHistory",
                "ecr:PutImageScanningConfiguration",
                "cloudformation:*",
                "iam:CreateRole",
                "iam:AttachRolePolicy",
                "ssm:GetParameters",
                "iam:PutRolePolicy",
```

```
                "ssm:GetParameter",
                "ssm:DeleteParameters",
                "ecr:DeleteRepository",
                "ssm:PutParameter",
                "ssm:DeleteParameter",
                "iam:PassRole",
                "ecr:SetRepositoryPolicy",
                "ssm:GetParametersByPath",
                "ecr:DescribeRepositories",
                "ecr:GetLifecyclePolicy"
            ],
            "Resource": [
                "arn:aws:ssm:aws-region:aws-account:parameter/cdk-bootstrap/*",
                "arn:aws:cloudformation:aws-region:aws-account:stack/CDKToolkit/*",
                "arn:aws:ecr:aws-region:aws-account:repository/cdk-*",
                "arn:aws:iam::aws-account:role/cdk-*"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "cloudformation:RegisterType",
                "cloudformation:CreateUploadBucket",
                "cloudformation:ListExports",
                "cloudformation:DescribeStackDriftDetectionStatus",
                "cloudformation:SetTypeDefaultVersion",
                "cloudformation:RegisterPublisher",
                "cloudformation:ActivateType",
                "cloudformation:ListTypes",
                "cloudformation:DeactivateType",
                "cloudformation:SetTypeConfiguration",
                "cloudformation:DeregisterType",
                "cloudformation:ListTypeRegistrations",
                "cloudformation:EstimateTemplateCost",
                "cloudformation:DescribeAccountLimits",
                "cloudformation:BatchDescribeTypeConfigurations",
                "cloudformation:CreateStackSet",
                "cloudformation:ListStacks",
                "cloudformation:DescribeType",
                "cloudformation:ListImports",
                "s3:*",
                "cloudformation:PublishType",
                "ecr:CreateRepository",
```

```
                "cloudformation:DescribePublisher",
                "cloudformation:DescribeTypeRegistration",
                "cloudformation:TestType",
                "cloudformation:ValidateTemplate",
                "cloudformation:ListTypeVersions"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> ロールを初めて使用する場合は、リソースポリシーステートメントで次のワイルドカードを使用し、使用可能になった後にリソース名でポリシーの範囲を絞り込みます。
>
> ```
> "Resource": "*"
> ```

- 次のカスタム信頼ポリシー：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                 ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

このロールがアカウント接続に追加されていることを確認します。アカウント接続への IAM ロールの追加の詳細については、「」を参照してください[アカウント接続への IAM ロールの追加](#)。

> **Note**
>
> 必要に応じて、ここでCodeCatalystWorkflowDevelopmentRole-*spaceName*ロールの名前を指定できます。このロールの詳細については、「アカウントとスペースのCodeCatalystWorkflowDevelopmentRole-***spaceName***ロールの作成」を参照してください。CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールには、セキュリティリスクをもたらす可能性のある非常に広範なアクセス許可があることを理解します。このロールは、セキュリティが懸念されないチュートリアルとシナリオでのみ使用することをお勧めします。

対応する UI: 設定タブ/'環境/アカウント/ロール'/ロール

Configuration

(*CDKBootstrapAction*/Configuration)

(必須)

アクションの設定プロパティを定義できるセクション。

対応する UI: 設定タブ

Region

(*CDKBootstrapAction*/Configuration/Region)

(必須)

ブートストラップスタックをデプロイ AWS リージョン する を指定します。このリージョンは、AWS CDK アプリがデプロイされているリージョンと一致する必要があります。リージョンコードの一覧については、「リージョンエンドポイント」を参照してください。

対応する UI: 設定タブ/リージョン

CdkCliVersion

(*CDKBootstrapAction*/Configuration/CdkCliVersion)

(オプション)

このプロパティは、AWS CDK デプロイアクションのバージョン 1.0.13 以降、およびAWS CDK ブートストラップアクションのバージョン 1.0.8 以降で使用できます。

次のいずれかを指定します。

- このアクションで使用する AWS Cloud Development Kit (AWS CDK) コマンドラインインターフェイス (CLI) (ツールキットとも呼ばれます AWS CDK) のフルバージョン。例えば、`2.102.1` などです。アプリケーションの構築とデプロイ時に一貫性と安定性を確保するために、フルバージョンを指定することを検討してください。

  または

- `latest`。 CDK CLI の最新の機能と修正を活用する`latest`には、 を指定することを検討してください。

アクションは、指定されたバージョンの AWS CDK CLI (または最新バージョン) を CodeCatalyst ビルドイメージ にダウンロードし、このバージョンを使用して CDK アプリケーションのデプロイまたは環境のブートストラップ AWS に必要なコマンドを実行します。

使用できるサポートされている CDK CLI バージョンのリストについては、AWS CDK 「バージョン」を参照してください。

このプロパティを省略すると、 アクションは、次のいずれかのトピックで説明されているデフォルトの AWS CDK CLI バージョンを使用します。

- 「デプロイAWS CDK 」アクションで使用される CDK CLI バージョン
- AWS CDK 「ブートストラップ」アクションで使用される CDK CLI バージョン

対応する UI: 設定タブ/AWS CDK CLI バージョン

ワークフローを使用して Amazon S3 にファイルを発行する

このセクションでは、 CodeCatalyst ワークフローを使用して Amazon S3 にファイルを発行する方法について説明します。これを行うには、Amazon S3 の公開アクションをワークフローに追加する必要があります。Amazon S3 パブリッシュアクションは、ソースディレクトリから Amazon S3 バケットにファイルをコピーします。ソースディレクトリは次の場所にあります。

- ソースリポジトリ 、、または
- 別のワークフローアクションによって生成された出力アーティファクト

Amazon S3 公開」アクションを使用するタイミング

以下の場合は、このアクションを使用します。

- Amazon S3 に保存するファイルを生成するワークフローがあります。

  例えば、Amazon S3 でホストする静的ウェブサイトを構築するワークフローがあるとします。この場合、ワークフローには、サイトの HTML およびサポートファイルを構築するためのビルドアクションと、ファイルを Amazon S3 にコピーするための Amazon S3 公開アクションが含まれます。 Amazon S3
- Amazon S3 に保存するファイルを含むソースリポジトリがあります。

  例えば、Amazon S3 に毎晩アーカイブするアプリケーションソースファイルを含むソースリポジトリがあるとします。

トピック

- Amazon S3 にファイルを発行するワークフローの例
- Amazon S3 公開」アクションの追加
- Amazon S3 公開」アクション YAML 定義

Amazon S3 にファイルを発行するワークフローの例

次のワークフローの例には、Amazon S3 の公開アクションとビルドアクションが含まれています。ワークフローは静的ドキュメントウェブサイトを構築し、Amazon S3 に公開してホストします。ワークフローは、順番に実行される次の構成要素で構成されます。

> **Note**
>
> 次のワークフロー例は説明を目的としており、追加の設定でのみ機能します。

- トリガー — このトリガーは、変更をソースリポジトリにプッシュすると、ワークフローの実行を自動的に開始します。トリガーについての詳細は、「トリガーを使用したワークフローの自動実行の開始」を参照してください。
- ビルドアクション (BuildDocs) – トリガー時に、アクションは静的ドキュメントウェブサイト (`mkdocs build`) を構築し、関連する HTML ファイルとサポートメタデータを というアーティ

ファクトに追加しますMyDocsSite。ビルドアクションの詳細については、「」を参照してくださいワークフローによる構築。

- Amazon S3 パブリッシュアクション (PublishToS3) – ビルドアクションが完了すると、このアクションはアーMyDocsSiteティファクト内のサイトをホスト用に Amazon S3 にコピーします。

```
Name: codecatalyst-s3-publish-workflow
SchemaVersion: 1.0

Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  BuildDocs:
    Identifier: aws/build@v1
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      Steps:
        - Run: echo BuildDocs started on `date`
        - Run: pip install --upgrade pip
        - Run: pip install mkdocs
        - Run: mkdocs build
        - Run: echo BuildDocs completed on `date`
    Outputs:
      Artifacts:
      - Name: MyDocsSite
        Files:
          - "site/**/*"

  PublishToS3:
    Identifier: aws/s3-publish@v1
    Environment:
      Name: codecatalyst-s3-publish-environment
      Connections:
        - Name: codecatalyst-account-connection
          Role: codecatalyst-s3-publish-build-role
    Inputs:
      Sources:
        - WorkflowSource
      Artifacts:
```

```
      - MyDocsSite
  Configuration:
    DestinationBucketName: my-bucket
    SourcePath: /artifacts/PublishToS3/MyDocSite/site
    TargetPath: my/docs/site
```

Amazon S3 公開」アクションの追加

次の手順を使用して、Amazon S3 の発行アクションをワークフローに追加します。

Visual

ビジュアルエディタを使用してAmazon S3 公開」アクションを追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. ビジュアル を選択します。
7. 左上で + Actions を選択してアクションカタログを開きます。
8. ドロップダウンリストから Amazon CodeCatalystを選択します。
9. Amazon S3 公開アクションを検索し、次のいずれかを実行します。
   - プラス記号 (+) を選択してワークフロー図にアクションを追加し、設定ペインを開きます。

     または
   - Amazon S3 パブリッシュ を選択します。アクションの詳細ダイアログボックスが表示されます。このダイアログボックスで、次の操作を行います。
     - (オプション) ソースを表示 を選択して、アクションのソースコード を表示します。
     - ワークフローに追加 を選択して、ワークフロー図にアクションを追加し、その設定ペインを開きます。
10. Inputs 、Configuration 、および Outputs タブで、必要に応じてフィールドに入力します。各フィールドの説明については、「」を参照してくださいAmazon S3 公開」アクション YAML

定義。このリファレンスでは、YAML エディタとビジュアルエディタの両方に表示される各フィールド (および対応する YAML プロパティ値) に関する詳細情報を提供します。

11. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
12. コミット を選択し、コミットメッセージを入力し、再度コミットを選択します。

YAML

YAML エディタを使用してAmazon S3 公開」アクションを追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. YAML を選択します。
7. 左上で + Actions を選択してアクションカタログを開きます。
8. ドロップダウンリストから Amazon CodeCatalystを選択します。
9. Amazon S3 公開アクションを検索し、次のいずれかを実行します。

   - プラス記号 (+) を選択してワークフロー図にアクションを追加し、その設定ペインを開きます。

     または

   - Amazon S3 パブリッシュ を選択します。アクションの詳細ダイアログボックスが表示されます。このダイアログボックスで、次の操作を行います。
     - (オプション) ソースを表示 を選択して、アクションのソースコード を表示します。
     - ワークフローに追加 を選択して、ワークフロー図にアクションを追加し、その設定ペインを開きます。

10. 必要に応じて YAML コードのプロパティを変更します。使用可能な各プロパティの説明は、「」に記載されていますAmazon S3 公開」アクション YAML 定義。

11. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
12. コミット を選択し、コミットメッセージを入力し、もう一度コミットを選択します。

Amazon S3 公開」アクション YAML 定義

Amazon S3 公開アクションの YAML 定義を次に示します。このアクションの使用方法については、「」を参照してくださいワークフローを使用して Amazon S3 にファイルを発行する。

このアクション定義は、より広範なワークフロー定義ファイル内のセクションとして存在します。ファイルの詳細については、「ワークフロー YAML 定義」を参照してください。

> **Note**
>
> 後続の YAML プロパティのほとんどには、対応する UI 要素がビジュアルエディタにあります。UI 要素を検索するには、Ctrl+F を使用します。 要素は、関連付けられた YAML プロパティとともに一覧表示されます。

```
# The workflow definition starts here.
# See ######### for details.

Name: MyWorkflow
SchemaVersion: 1.0
Actions:

# The action definition starts here.
  S3Publish_nn:
    Identifier: aws/s3-publish@v1
    DependsOn:
      - build-action
    Compute:
      Type: EC2 | Lambda
      Fleet: fleet-name
    Timeout: timeout-minutes
    Inputs:
      Sources:
        - source-name-1
      Artifacts:
        - artifact-name
      Variables:
```

```
        - Name: variable-name-1
          Value: variable-value-1
        - Name: variable-name-2
          Value: variable-value-2
    Environment:
      Name: environment-name
      Connections:
        - Name: account-connection-name
          Role: iam-role-name
    Configuration:
      SourcePath: my/source
      DestinationBucketName: s3-bucket-name
      TargetPath: my/target
```

S3Publish

(必須)

アクションの名前を指定します。すべてのアクション名は、ワークフロー内で一意である必要があります。アクション名は、英数字 (a～z、A～Z、0～9)、ハイフン (-)、アンダースコア (_) に制限されています。スペースは使用できません。引用符を使用してアクション名で特殊文字やスペースを有効にすることはできません。

デフォルト: S3Publish_nn。

対応する UI: 設定タブ/アクション名

Identifier

(*S3Publish*/Identifier)

(必須)

アクションを識別します。バージョンを変更しない限り、このプロパティを変更しないでください。詳細については、「アクションのメジャー、マイナー、またはパッチバージョンの指定」を参照してください。

デフォルト: aws/s3-publish@v1。

対応する UI: ワークフロー図/S3Publish_nn/aws/s3-publish@v1 ラベル

DependsOn

(*S3Publish*/DependsOn)

(オプション)

このアクションを実行するために正常に実行する必要があるアクション、アクショングループ、またはゲートを指定します。

「依存」機能の詳細については、「」を参照してください他のアクションに依存するようにアクションを設定する。

対応する UI: の入力タブ/依存 - オプション

Compute

(*S3Publish*/Compute)

(オプション)

ワークフローアクションを実行するために使用されるコンピューティングエンジン。コンピューティングはワークフローレベルまたはアクションレベルで指定できますが、両方を指定することはできません。ワークフローレベルで指定すると、コンピューティング設定はワークフローで定義されたすべてのアクションに適用されます。ワークフローレベルでは、同じインスタンスで複数のアクションを実行することもできます。詳細については、「アクション間でのコンピューティングの共有」を参照してください。

対応する UI: なし

Type

(*S3Publish*/Compute/Type)

(Computeが含まれている場合は必須)

コンピューティングエンジンのタイプ。次のいずれかの値を使用できます。

- EC2 (ビジュアルエディタ) または EC2 (YAML エディタ)

  アクション実行中の柔軟性のために最適化されました。
- Lambda (ビジュアルエディタ) または Lambda (YAML エディタ)

  アクションの起動速度を最適化しました。

コンピューティングタイプの詳細については、「コンピューティングタイプ」を参照してください。

対応する UI: 設定タブ/コンピューティングタイプ

Fleet

(*S3Publish*/Compute/Fleet)

(オプション)

ワークフローまたはワークフローアクションを実行するマシンまたはフリートを指定します。オンデマンドフリートでは、アクションが開始されると、ワークフローは必要なリソースをプロビジョニングし、アクションが終了するとマシンは破棄されます。オンデマンドフリートの例: `Linux.x86-64.Large`、`Linux.x86-64.XLarge`。オンデマンドフリートの詳細については、「」を参照してくださいオンデマンドフリートのプロパティ。

プロビジョニングされたフリートでは、ワークフローアクションを実行するように一連の専用マシンを設定します。これらのマシンはアイドル状態のままで、すぐにアクションを処理できます。プロビジョニングされたフリートの詳細については、「」を参照してくださいプロビジョニングされたフリートのプロパティ。

`Fleet` を省略した場合、デフォルトは ですLinux.x86-64.Large。

対応する UI: 設定タブ/コンピューティングフリート

Timeout

(*S3Publish*/Timeout)

(必須)

がアクション CodeCatalyst を終了するまでにアクションを実行できる時間を分単位で指定します (YAML エディタ)、または時間と分単位で指定します (ビジュアルエディタ)。最小値は 5 分で、最大値は「」で説明されていますワークフローのクォータ。デフォルトのタイムアウトは、最大タイムアウトと同じです。

対応する UI: 設定タブ/タイムアウト - オプション

Inputs

(*S3Publish*/Inputs)

(オプション)

`Inputs` セクションでは、ワークフローの実行中に がS3Publish必要とするデータを定義します。

> **Note**
>
> AWS CDK デプロイアクションごとに最大 4 つの入力 (1 つのソースと 3 つのアーティファクト) が許可されます。変数はこの合計にはカウントされません。

異なる入力 (ソースとアーティファクトなど) にあるファイルを参照する必要がある場合、ソース入力はプライマリ入力、アーティファクトはセカンダリ入力です。セカンダリ入力内のファイルへの参照は、プライマリから区別するために特別なプレフィックスが付けられます。詳細については、「例: 複数のアーティファクト内のファイルを参照する」を参照してください。

対応する UI: Inputs タブ

Sources

(*S3Publish*/Inputs/Sources)

(Amazon S3 に発行するファイルがソースリポジトリに保存されている場合に必要です)

Amazon S3 に発行するファイルがソースリポジトリに保存されている場合は、そのソースリポジトリのラベルを指定します。現在、サポートされているラベルは のみです`WorkflowSource`。

Amazon S3 に公開するファイルがソースリポジトリに含まれていない場合は、別のアクションによって生成されたアーティファクトに存在する必要があります。

sources の詳細については、「ワークフローをソースリポジトリに接続する」を参照してください。

対応する UI: 入力タブ/ソース - オプション

Artifacts - input

(*S3Publish*/Inputs/Artifacts)

(Amazon S3 に発行するファイルが、前のアクションの出力アーティファクトに保存されている場合に必要です)

Amazon S3 に公開するファイルが、前のアクションによって生成されたアーティファクトに含まれている場合は、ここでそのアーティファクトを指定します。ファイルがアーティファクトに含まれていない場合は、ソースリポジトリに存在する必要があります。

アーティファクトの詳細については、「」を参照してくださいアーティファクトを使用したワークフロー内のアクション間のデータの共有。

対応する UI: 設定タブ/アーティファクト - オプション

Variables - input

(*S3Publish*/Inputs/Variables)

(オプション)

アクションで使用できるようにしたい入力変数を定義する名前と値のペアのシーケンスを指定します。変数名は、英数字 (a～z、A～Z、0～9)、ハイフン (-)、アンダースコア (_) に制限されています。スペースは使用できません。変数名で特殊文字やスペースを有効にするために引用符を使用することはできません。

例を含む変数の詳細については、「」を参照してくださいワークフローでの変数の設定と使用。

対応する UI: 入力タブ/変数 - オプション

Environment

(*S3Publish*/Environment)

(必須)

アクションで使用する CodeCatalyst 環境を指定します。

環境の詳細については、環境を使用して AWS アカウント および VPCsにデプロイする CodeCatalyst「」および「」を参照してください環境を作成する。

対応する UI: 設定タブ/'環境/接続/ロール'/環境

Name

(*S3Publish*/Environment/Name)

(Environmentが含まれている場合は必須)

アクションに関連付ける既存の環境の名前を指定します。

対応する UI: 設定タブ/'環境/接続/ロール'/環境

Connections

(*S3Publish*/Environment/Connections)

(Environmentが含まれている場合は必須)

アクションに関連付けるアカウント接続を指定します。で最大 1 つのアカウント接続を指定できますEnvironment。

アカウント接続の詳細については、「」を参照してください接続された AWS リソースへのアクセスを許可する AWS アカウント。アカウント接続を環境に関連付ける方法については、「」を参照してください環境を作成する。

対応する UI: 設定タブ/'環境/接続/ロール'/接続

Name

(*S3Publish*/Environment/Connections/Name)

(必須)

アカウント接続の名前を指定します。

対応する UI: 設定タブ/'環境/接続/ロール'/接続

Role

(*S3Publish*/Environment/Connections/Role)

(必須)

Amazon S3 公開アクションが Amazon S3 へのアクセス AWS とファイルのコピーに使用する IAM Amazon S3ロールの名前を指定します。このロールに以下が含まれていることを確認してください。

- 次のアクセス許可ポリシー：

> **Warning**
> アクセス許可を次のポリシーに示すものに制限します。より広範なアクセス許可を持つロールを使用すると、セキュリティ上のリスクが生じる可能性があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:ListBucket",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::bucket-name",
                "arn:aws:s3:::bucket-name/*"
            ]
        }
    ]
}
```

- 次のカスタム信頼ポリシー：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                 ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

このロールがアカウント接続に関連付けられていることを確認します。IAM ロールとアカウント接続の関連付けの詳細については、「」を参照してくださいアカウント接続への IAM ロールの追加。

> **Note**
>
> 必要に応じて、ここでCodeCatalystWorkflowDevelopmentRole-*spaceName*ロールの名前を指定できます。このロールの詳細については、「アカウントとスペース

のCodeCatalystWorkflowDevelopmentRole-*spaceName*ロールの作成」を参照してください。CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールには、セキュリティリスクをもたらす可能性のある非常に広範なアクセス許可があることを理解します。このロールは、セキュリティが懸念されないチュートリアルやシナリオでのみ使用することをお勧めします。

対応する UI: 設定タブ/'環境/接続/ロール'/ロール

Configuration

(*S3Publish*/Configuration)

(必須)

アクションの設定プロパティを定義できるセクション。

対応する UI: 設定タブ

SourcePath

(*S3Publish*/Configuration/SourcePath)

(必須)

Amazon S3 に発行するディレクトリまたはファイルの名前とパスを指定します。ディレクトリまたはファイルは、ソースリポジトリまたは前のアクションのアーティファクトに存在し、ソースリポジトリまたはアーティファクトルートを基準にしています。

例:

を指定すると、 の内容/myFolderが Amazon S3 ./myFolder/にコピーされ、基になるディレクトリ構造が保持されます。

Amazon S3 へのmyfile.txt./myFolder/myfile.txtコピーのみを指定します。(ディレクトリ構造は削除されます)。

ワイルドカードは使用できません。

> **Note**
>
> ディレクトリまたはファイルパスにプレフィックスを追加して、検索するアーティファクトまたはソースを示す必要がある場合があります。詳細については、「ソースリポジトリ内の

ファイルを参照する」および「アーティファクト内のファイルを参照する」を参照してください。

対応する UI: 設定タブ/ソースパス

DestinationBucketName

(*S3Publish*/Configuration/DestinationBucketName)

(必須)

ファイルを発行する Amazon S3 バケットの名前を指定します。

対応する UI: 設定タブ/送信先バケット - オプション

TargetPath

(*S3Publish*/Configuration/TargetPath)

(オプション)

ファイルを発行する Amazon S3 のディレクトリの名前とパスを指定します。ディレクトリが存在しない場合は、作成されます。ディレクトリパスにバケット名を含めることはできません。

例:

```
myS3Folder
```

```
./myS3Folder/myS3Subfolder
```

対応する UI: 設定タブ/送信先ディレクトリ - オプション

環境を使用して AWS アカウント および VPCsにデプロイする CodeCatalyst

ワークフロー CodeCatalyst アクションを使用して、アプリケーションやその他のリソースを AWS アカウント または Amazon VPC にデプロイできます。これを実現するには、 CodeCatalyst 環境を設定する必要があります。

開発環境 と混同される環境 は、コードがデプロイされる場所です。通常、実行中のアプリケーションのインスタンスと関連するインフラストラクチャが含まれます。開発、テスト、ステージング、本番稼働などの名前を環境に付けます。によって生成された環境 CodeCatalyst へのデプロ

イは、環境ページに表示されます。環境を設定するには、 などの名前を付けmy-production-environment、それを に関連付けます AWS アカウント。

環境は、デプロイ情報の表示に加えて、ワークフローアクション に AWS IAM ロールを割り当てるメカニズムとしても機能します。

1 つのワークフロー内に複数の環境が存在することはできますか？

はい。ワークフローに複数のアクションが含まれている場合、それらの各アクションに環境を割り当てることができます。例えば、2 つのデプロイアクションを含むワークフローがあるとします。1 つはmy-staging-enviroment環境を割り当て、もう 1 つはmy-production-environment環境を割り当てます。

環境をサポートするアクションはどれですか？

以下のアクションでは、デプロイ情報を環境ページに表示することができます。

- AWS CloudFormation スタックのデプロイ – 詳細については、「」を参照してください。 ワークフローを使用した AWS CloudFormation スタックのデプロイ
- Amazon ECS にデプロイする – 詳細については、「」を参照してください。 ワークフローを使用した Amazon Elastic Container Service (ECS) へのアプリケーションのデプロイ
- Kubernetes クラスターにデプロイする – 詳細については、「」を参照してください。 ワークフローを使用した Amazon Elastic Kubernetes Service へのアプリケーションのデプロイ
- AWS CDK デプロイ – 詳細については、「」を参照してください。 ワークフローを使用した AWS Cloud Development Kit (AWS CDK) アプリケーションのデプロイ

> **Note**
>
> AWS アカウントで アクションにアクセスしてオペレーションを実行することを許可する場合は、そのアクションを環境に関連付ける必要があります。多くのアクションは環境の関連付けをサポートしています。これには、前述のアクションが含まれますが、これらに限定されません。環境の関連付けをサポートするアクションは、ビジュアルエディタ の設定タブに環境ドロップダウンリストが含まれるため、わかります。

サポートされるリージョン

環境ページには、任意の AWS リージョンのリソースを表示できます。

環境は必須ですか？

環境は、割り当てられたワークフローアクションがリソースを AWS クラウドにデプロイする場合や、その他の理由 (モニタリングやレポートなど) で AWS サービスと通信する場合に必須です。

トピック

- 環境を作成する
- 環境、アカウント接続、IAM ロールをワークフローアクションに関連付ける
- VPC 接続を環境に関連付ける
- 環境 AWS アカウント への の関連付け

環境を作成する

次の手順を使用して、後でワークフローアクションに関連付けることができる空の環境を作成します。

開始する前に

以下が必要です。

- CodeCatalyst スペース。詳細については、「のセットアップと へのサインイン CodeCatalyst」を参照してください。
- CodeCatalyst プロジェクト。詳細については、「設計図を使用したプロジェクトの作成」を参照してください。
- ワークフローアクションが にアクセスするために必要な IAM ロールを含む AWS アカウント接続 AWS。環境ごとに最大 1 つのアカウント接続を使用できます。詳細については、「接続された AWS リソースへのアクセスを許可する AWS アカウント」を参照してください。

> **Note**
>
> アカウント接続なしで環境を作成できますが、後で接続に戻って追加する必要があります。

環境を作成するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。

3. ナビゲーションペインで CI/CD を選択し、環境 を選択します。
4. 環境名 に、 **Production**や などの名前を入力しますStaging。
5. 環境タイプ で、次のいずれかを選択します。

   - 非本番環境 – アプリケーションをテストして、本番環境に移行する前に意図したとおりに動作していることを確認することができる環境。
   - 本番稼働 – 公開されており、確定したアプリケーションをホストする「ライブ」環境。

     本番稼働用 を選択すると、環境が関連付けられているアクションの横に本番稼働用バッジが表示されます。このバッジは、どのアクションが本番環境にデプロイされているかをすばやく確認するのに役立ちます。バッジの外観以外には、本番環境と非本番環境の間に違いはありません。
6. (オプション) VPC 接続 で、この環境に関連付ける VPC 接続を選択します。この VPC 接続の作成の詳細については、「 管理者ガイド」の「Amazon Virtual Private Cloud の管理」を参照してください。 CodeCatalyst
7. (オプション) 説明 に、 などの説明を入力しますProduction environment for the hello-world app。
8. 接続 - オプションで、この環境に関連付ける AWS アカウント接続を選択します。アカウント接続に、環境に関連付ける IAM ロールが含まれていることを確認します。この接続の作成の詳細については、「」を参照してください接続された AWS リソースへのアクセスを許可する AWS アカウント。
9. 環境の作成 を選択します。空の環境 CodeCatalyst を作成します。

次のステップ

- 環境を作成したので、ワークフローアクションに関連付ける準備が整いました。詳細については、「環境、アカウント接続、IAM ロールをワークフローアクションに関連付ける」を参照してください。

環境、アカウント接続、IAM ロールをワークフローアクションに関連付ける

環境、アカウント接続、および IAM ロールをサポートされているワークフローアクション に関連付けると、IAM ロールがアクションで使用できるようになります。IAM ロールへのアクセスに加えて、 アクションのデプロイ情報も環境ページにインポートされる場合があります。詳細については、「環境をサポートするアクションはどれですか？」を参照してください。

環境、アカウント接続、IAM ロールを アクションに関連付けるには、以下の手順に従います。

ステップ 1: 環境、アカウント接続、ロールをワークフローアクションに関連付ける

環境、アカウント接続、およびロールをワークフローアクションに関連付けるには、次の手順を使用します。

Visual

ビジュアルエディタを使用して環境、アカウント接続、ロールをワークフローアクションに関連付けるには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. ビジュアル を選択します。
7. ワークフロー図で、 環境でサポートされているアクションを選択します。詳細については、「環境をサポートするアクションはどれですか？」を参照してください。
8. 設定タブを選択し、次のようにフィールドに情報を指定します。

   環境

   アクションで使用する CodeCatalyst 環境を指定します。

   環境の詳細については、環境を使用して AWS アカウント および VPCsにデプロイする CodeCatalyst 「」および「」を参照してください環境を作成する。

   アカウント接続または接続 - オプション (いずれかが利用可能)

   アクションに関連付けるアカウント接続を指定します。で最大 1 つのアカウント接続を指定できますEnvironment。

   アカウント接続の詳細については、「」を参照してください接続された AWS リソースへのアクセスを許可する AWS アカウント。アカウント接続を環境に関連付ける方法については、「」を参照してください環境を作成する。

ロール

このアクションが Amazon S3 や Amazon ECR などの AWS サービスにアクセスして操作するために使用する IAM ロールの名前を指定します。 Amazon S3 このロールがアカウント接続に追加されていることを確認します。アカウント接続に IAM ロールを追加するには、「」を参照してくださいアカウント接続への IAM ロールの追加。

> **Note**
>
> 十分なアクセス許可があれば、ここでCodeCatalystWorkflowDevelopmentRole-*spaceName*ロールの名前を指定できます。このロールの詳細については、「アカウントとスペースのCodeCatalystWorkflowDevelopmentRole-**spaceName**ロールの作成」を参照してください。CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールには、セキュリティリスクをもたらす可能性のある非常に広範なアクセス許可があることを理解します。このロールは、セキュリティが懸念されないチュートリアルやシナリオでのみ使用することをお勧めします。

ロールがリストに表示されない場合は、そのロールをアカウント接続に関連付けていないためです。詳細については、「アカウント接続への IAM ロールの追加」を参照してください。

9. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
10. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

YAML

YAML エディタを使用して環境、アカウント接続、ロールをワークフローアクションに関連付けるには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。

5. [編集] を選択します。
6. YAML を選択します。
7. 環境に関連付けるワークフローアクションで、次のようなコードを追加します。

```
action-name
  Environment:
    Name: environment-name
    Connections:
      - Name: account-connection-name
        Role: iam-role-name
```

詳細については、アクションのワークフロー YAML 定義「」を参照してください。

8. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
9. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

ステップ 2: 環境を入力する

環境、アカウント接続、ロールをワークフローアクションに関連付けると、環境ページにデプロイ情報を入力できます。以下の手順に従って、環境ページにデータを入力します。

> **Note**
>
> 環境ページは、ワークフローアクションのサブセットでのみサポートされています。詳細については、「環境をサポートするアクションはどれですか？」を参照してください。

環境にデータを入力するには

1. で変更をコミットしたときにワークフロー実行が自動的に開始されない場合はステップ 1: 環境、アカウント接続、ロールをワークフローアクションに関連付ける、次のように手動で実行を開始します。

   a. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。

   b. 実行を開始するワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。

c. [実行] を選択します。

ワークフロー実行は新しいデプロイを開始します。これにより、 CodeCatalyst は 環境 の下にアプリケーションリソース情報を追加します。

2. アプリケーションリソースが環境の下に表示されることを確認します。

a. ナビゲーションペインで CI/CD を選択し、環境 を選択します。

b. 環境を選択します (例: `Production`) 。

c. デプロイアクティビティタブを選択し、デプロイのステータスが SUCCEEDED になっていることを確認します。これは、ワークフローの実行がアプリケーションリソースを正常にデプロイしたことを示します。

d. デプロイターゲットタブを選択し、アプリケーションリソースが表示されることを確認します。

VPC 接続を環境に関連付ける

アクションが VPC 接続を持つ環境で設定されている場合、アクションは VPC に接続して実行され、ネットワークルールに準拠し、関連付けられた VPC で指定されたリソースにアクセスします。1 つ以上の環境で同じ VPC 接続を使用できます。

VPC 接続を環境に関連付けるには、以下の手順に従います。

VPC 接続を環境に関連付けるには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、環境 を選択します。
4. 環境を選択します (例: `Production`) 。
5. 環境プロパティタブを選択します。
6. VPC 接続の管理 を選択し、目的の VPC 接続を選択し、確認 を選択します。これにより、選択した VPC 接続がこの環境に関連付けられます。

詳細については、「 CodeCatalyst 管理者ガイド」の「Amazon Virtual Private Cloud の管理」を参照してください。

環境 AWS アカウント への の関連付け

を環境に関連付けるには AWS アカウント 、以下の手順に従います。

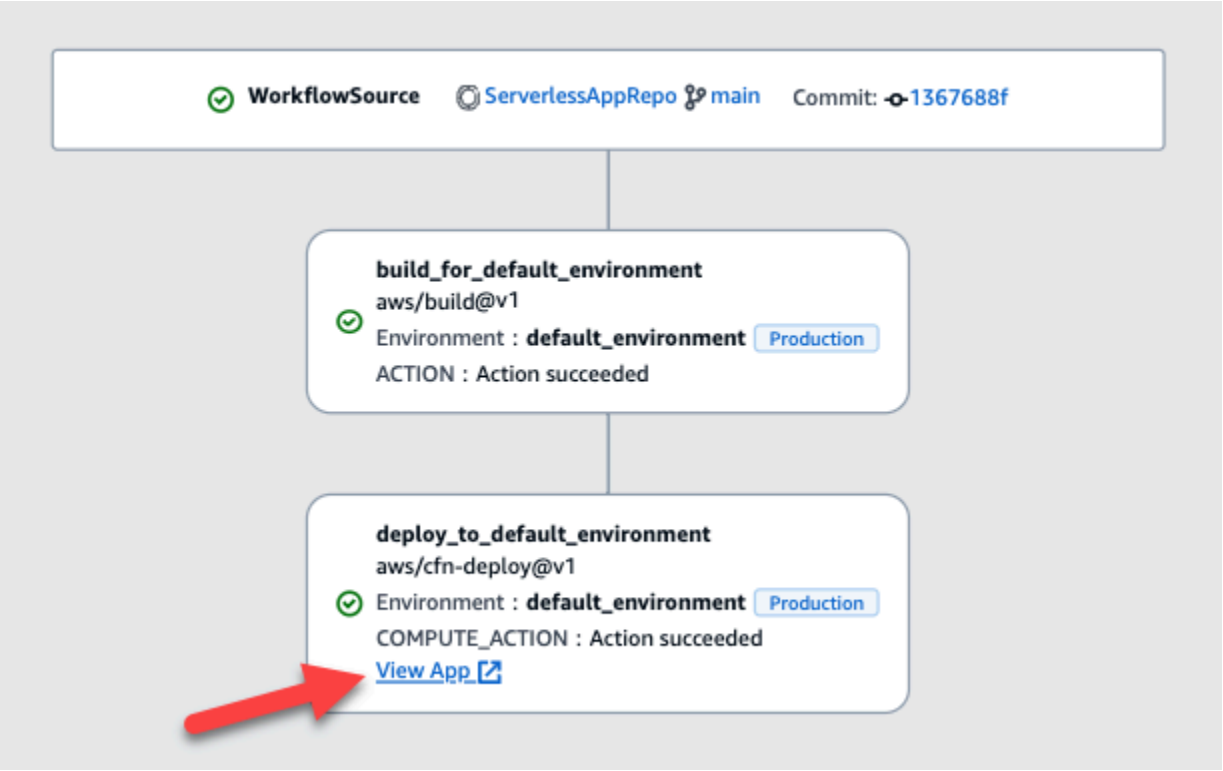
AWS アカウント を環境に関連付けるには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、環境 を選択します。
4. 環境を選択します (例: `Production`) 。
5. 環境プロパティタブを選択します。
6. 関連付け AWS アカウント を選択し、目的の を選択し AWS アカウント、関連付け を選択します。これにより、選択した がこの環境 AWS アカウント に関連付けられます。

詳細については、「接続された AWS リソースへのアクセスを許可する AWS アカウント」を参照してください。

ワークフロー図にデプロイされたアプリケーションの URL を表示する

ワークフローがアプリケーションをデプロイする場合、アプリケーションの URL をクリック可能なリンクとして表示 CodeCatalyst するように Amazon を設定できます。このリンクは、CodeCatalyst コンソールのデプロイしたアクション内に表示されます。次のワークフロー図は、アクションの下部に表示されるアプリケーション URL の表示を示しています。



この URL を CodeCatalyst コンソールでクリック可能にすることで、アプリケーションのデプロイをすばやく検証できます。

> **Note**
>
> アプリ URL は、Amazon ECS へのデプロイアクションではサポートされていません。

この機能を有効にするには、、、appurl、または を含む名前の出力変数をアクションに追加しますendpointurl。名前は、結合ダッシュ (-) 、アンダースコア ()、またはスペース (_) の有無にかかわらず使用できます 。文字列では大文字と小文字は区別されません。変数の値を、デプロイされたアプリケーションの httpまたは https URL に設定します。

> **Note**
>
> 既存の出力変数を更新して app url、、または endpoint url文字列を含める場合は、この変数へのすべての参照を更新して新しい変数名を使用します。

詳細な手順については、次のいずれかの手順を参照してください。

- 「デプロイAWS CDK 」アクションでアプリケーション URL を表示するには
- AWS CloudFormation 「スタックのデプロイ」アクションにアプリケーション URL を表示するには
- 他のすべてのアクションでアプリケーション URL を表示するには

URL の設定が完了したら、次の手順に従って URL が期待どおりに表示されることを確認します。

- アプリケーション URL が追加されたことを確認するには

「デプロイAWS CDK 」アクションでアプリケーション URL を表示するには

1. AWS CDK デプロイアクションを使用している場合は、 AWS CDK アプリケーションコードにCfnOutputコンストラクト (キーと値のペア) を追加します。

   - キー名にはappurl、ダッシュ ()、アンダースコア (-) endpointurl、またはスペース (_) が結合されているかどうかにかかわらず、、、または が含まれている必要があります 。文字列では大文字と小文字は区別されません。
   - 値は、デプロイされたアプリケーションの httpまたは https URL である必要があります。

   例えば、 AWS CDK コードは次のようになります。

```
import { Duration, Stack, StackProps, CfnOutput, RemovalPolicy} from 'aws-cdk-lib';
import * as dynamodb from 'aws-cdk-lib/aws-dynamodb';
import * as s3 from 'aws-cdk-lib/aws-s3';
import { Construct } from 'constructs';
import * as cdk from 'aws-cdk-lib';
export class HelloCdkStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);
    const bucket = new s3.Bucket(this, 'my-bucket', {
      removalPolicy: RemovalPolicy.DESTROY,
    });
    new CfnOutput(this, 'APP-URL', {
      value: https://mycompany.myapp.com,
      description: 'The URL of the deployed application',
      exportName: 'myApp',
    });
    ...
  }
```

```
}
```

CfnOutput コンストラクトの詳細については、AWS Cloud Development Kit (AWS CDK) 「 API リファレンス」の「 インターフェイス CfnOutputProps」を参照してください。

2. コードを保存してコミットします。
3. アプリケーション URL が追加されたことを確認するには に進みます。

AWS CloudFormation 「スタックのデプロイ」アクションにアプリケーション URL を表示するには

1. AWS CloudFormation スタックのデプロイアクションを使用している場合は、テンプレートまたは CloudFormation テンプレートの Outputsセクションに次の特性 AWS SAM を持つ出力を追加します。

   - キー (論理 ID とも呼ばれます) にはappurl、ダッシュ ()、アンダースコア ()endpointurl、またはスペース (-) を結合するかどうかにかかわらず、_、、または が含まれている必要があります 。文字列では大文字と小文字は区別されません。
   - 値は、デプロイされたアプリケーションの httpまたは https URL である必要があります。

   例えば、 CloudFormation テンプレートは次のようになります。

```
"Outputs" : {
  "APP-URL" : {
    "Description" : "The URL of the deployed app",
    "Value" : "https://mycompany.myapp.com",
    "Export" : {
      "Name" : "My App"
    }
  }
}
```

   CloudFormation 出力の詳細については、「 AWS CloudFormation ユーザーガイド」の「出力」を参照してください。

2. コードを保存してコミットします。
3. アプリケーション URL が追加されたことを確認するには に進みます。

他のすべてのアクションでアプリケーション URL を表示するには

ビルドアクション やGitHub アクション など、別のアクションを使用してアプリケーションをデプロイする場合は、次の操作を実行してアプリケーション URL を表示します。

1. ワークフロー定義ファイルのアクションの Inputsまたは Stepsセクションで環境変数を定義します。変数には次の特性が必要です。

   - にはappurl、結合ダッシュ ()、アンダースコア (-) endpointurl、またはスペース (_) の有無にかかわらず、、、または が含まれているname必要があります 。文字列では大文字と小文字は区別されません。
   - 値は、デプロイされたアプリケーションの httpまたは https URL である必要があります。

   例えば、ビルドアクションは次のようになります。

   ```
   Build-action:
     Identifier: aws/build@v1
     Inputs:
       Variables:
         - Name: APP-URL
           Value: https://mycompany.myapp.com
   ```

   ... またはこれ :

   ```
   Actions:
     Build:
       Identifier: aws/build@v1
       Configuration:
         Steps:
           - Run: APP-URL=https://mycompany.myapp.com
   ```

   環境変数の定義の詳細については、「」を参照してください変数の定義。

2. 変数をエクスポートします。

   例えば、ビルドアクションは次のようになります。

   ```
   Build-action:
     ...
     Outputs:
       Variables:
   ```

```
      - APP-URL
```

変数のエクスポートについては、「」を参照してください[他のアクションで使用できるように変数をエクスポートする]。

3. (オプション) Validate を選択して、コミットする前にワークフローの YAML コードを検証します。
4. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。
5. [アプリケーション URL が追加されたことを確認するには] に進みます。

アプリケーション URL が追加されたことを確認するには

- 自動的に開始されていない場合は、ワークフロー実行を開始します。新しい実行では、アプリケーション URL がクリック可能なリンクとしてワークフロー図に表示される必要があります。実行の開始の詳細については、「」を参照してください[ワークフローの手動実行の開始]。

デプロイターゲットの削除

CodeCatalyst コンソールの環境ページから Amazon ECS クラスターや AWS CloudFormation スタックなどのデプロイターゲットを削除できます。

> **Important**
>
> デプロイターゲットを削除すると、 CodeCatalyst コンソールから削除されますが、それをホストする AWS サービスで引き続き使用できます (まだ存在する場合) 。

でターゲットが古くなった場合は、デプロイターゲットを削除することを検討してください CodeCatalyst。次の場合、ターゲットは古くなる可能性があります。

- ターゲットにデプロイされたワークフローを削除しました。
- デプロイ先のスタックまたはクラスターを変更しました。
- コンソールの または Amazon ECS サービスからスタック CloudFormation またはクラスターを削除しました AWS 。

デプロイターゲットを削除するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、環境 を選択します。
4. 削除するデプロイターゲットを含む環境の名前を選択します。環境の詳細については、「」を参照してください環境を使用して AWS アカウント および VPCsにデプロイする CodeCatalyst 。
5. デプロイターゲットタブを選択します。
6. 削除するデプロイターゲットの横にあるラジオボタンを選択します。
7. [削除] を選択します。

   ターゲットはページから削除されます。

コミットによるデプロイステータスの追跡

開発ライフサイクルのどの時点でも、バグ修正、新機能、その他の影響のある変更など、特定のコミットのデプロイステータスを把握することが重要です。デプロイステータス追跡機能が開発チームに役立つ次のシナリオを考えてみましょう。

- 開発者として、バグに対処する修正を行い、チームのデプロイ環境全体のデプロイのステータスを報告したいと考えています。
- リリースマネージャーとして、デプロイされたコミットのリストを表示して、そのデプロイステータスを追跡およびレポートします。

CodeCatalyst は、個々のコミットまたは変更がデプロイされた場所と環境を一目で判断するために使用できるビューを提供します。このビューには以下が含まれます。

- コミットのリスト。
- コミットを含むデプロイのステータス。
- コミットが正常にデプロイされる環境。
- CI/CD ワークフローのコミットに対して実行されるテストのステータス。

次の手順では、このビューに移動して使用してプロジェクトの変更を追跡する方法について詳しく説明します。

> **Note**
>
> コミットによるデプロイステータスの追跡は、CodeCatalyst リポジトリ でのみサポートされます。この機能は、 GitHub または Bitbucket リポジトリ では使用できません。

コミットでデプロイステータスを追跡するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、追跡の変更 を選択します。
4. メインペインの上部にある 2 つのドロップダウンリストで、リリースステータスを表示するコミットを含むソースリポジトリとブランチを選択します。
5. 変更の表示 を選択します。

   コミットのリストが表示されます。

   コミットごとに、以下を表示できます。

   - ID、作成者、メッセージ、コミット日時などの情報をコミットします。詳細については、「でソースリポジトリを使用してコードを保存し、共同作業する CodeCatalyst」を参照してください。
   - 各環境へのデプロイのステータス。詳細については、「環境を使用して AWS アカウント および VPCsにデプロイする CodeCatalyst 」を参照してください。
   - テストとコードカバレッジの結果。詳細については、「ワークフローを使用したテスト」を参照してください。

   > **Note**
   >
   > Software Composition Analysis (SCA) の結果は表示されません。

6. (オプション) 最新のデプロイ、詳細なコードカバレッジ、ユニットテスト情報など、特定のコミットに関連する変更に関する詳細を表示するには、そのコミットの詳細を表示を選択します。

デプロイログの表示

特定のデプロイアクションに関連するログを表示して、Amazon の問題のトラブルシューティングを行うことができます CodeCatalyst。

ワークフロー または環境 ???からログを表示できます。

ワークフローから開始するデプロイアクションのログを表示するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. 実行 を選択します。
6. アプリケーションをデプロイしたワークフロー実行を選択します。
7. ワークフロー図で、ログを表示するアクションを選択します。
8. ログタブを選択し、セクションを展開してログメッセージを表示します。
9. その他のログを表示するには、概要タブを選択し、「表示 CloudFormation」 (使用可能な場合) を選択して、さらに多くのログを表示します。にサインインする必要がある場合があります AWS。

環境から開始するデプロイアクションのログを表示するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、環境 を選択します。
4. アプリケーションがデプロイされた環境を選択します。
5. デプロイアクティビティ で、ワークフロー実行 ID 列を見つけ、スタックをデプロイしたワークフロー実行を選択します。
6. ワークフロー図で、ログを表示するアクションを選択します。
7. ログタブを選択し、セクションを展開してログメッセージを表示します。
8. その他のログを表示するには、概要タブを選択し、「表示 CloudFormation」 (使用可能な場合) を選択して、さらに多くのログを表示します。にサインインする必要がある場合があります AWS。

デプロイステータス、コミット、プルリクエストの表示

Amazon でのデプロイに関する以下の情報を表示できます CodeCatalyst。

- デプロイステータス、開始時刻、終了時刻、履歴、イベントの所要時間などのデプロイアクティビティ。
- スタック名 AWS リージョン、最終更新時刻、および関連するワークフロー。
- リクエストをコミットしてプルします。
- CloudFormation イベントや出力などのアクション固有の情報。

ワークフロー 、環境 ???、またはワークフローアクション からデプロイ情報を表示できます。

ワークフローから開始するデプロイ情報を表示するには

- アプリケーションをデプロイしたワークフロー実行に移動します。手順については、「ワークフローの実行ステータスと詳細の表示」を参照してください。

環境から開始するデプロイ情報を表示するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、環境 を選択します。
4. スタックがデプロイされた環境を選択します。例: `Production`。
5. デプロイアクティビティを選択して、スタックのデプロイ履歴、デプロイのステータス (SUCCEEDED や FAILED など)、およびその他のデプロイ関連情報を表示します。
6. デプロイターゲットを選択すると、環境にデプロイされたスタック、クラスター、またはその他のターゲットに関する情報が表示されます。スタック名、リージョン、プロバイダー、識別子などの情報を表示できます。

アクションから開始するデプロイ情報を表示するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。

4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. ワークフロー図で、アプリケーションをデプロイしたワークフローアクションを選択します。例えば、 を選択できますDeployCloudFormationStack。
6. 右側のペインの内容を確認して、アクション固有のデプロイ情報を確認します。

ワークフローの作成

ワークフローは、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) システムの一部としてコードを構築、テスト、デプロイする方法を説明する自動化された手順です。ワークフローは、ワークフローの実行中に実行する一連のステップまたはアクションを定義します。ワークフローは、ワークフローを開始するイベント、またはトリガー も定義します。ワークフローを設定するには、CodeCatalyst コンソールのビジュアルまたは YAML エディタ を使用してワークフロー定義ファイルを作成します。

> **Tip**
>
> プロジェクトでワークフローを使用する方法を簡単に確認するには、設計図 を使用してプロジェクトを作成します。各ブループリントは、レビュー、実行、および実験できる機能するワークフローをデプロイします。

でワークフローを作成するには、次の手順を使用します CodeCatalyst。

ワークフローの詳細については、「でワークフローを使用して構築、テスト、デプロイする CodeCatalyst」を参照してください。

Visual

ビジュアルエディタを使用してワークフローを作成するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの作成 を選択します。

   ワークフローの作成ダイアログボックスが表示されます。

5. ソースリポジトリ フィールドで、ワークフロー定義ファイルが存在するソースリポジトリを選択します。ファイルは、選択したリポジトリの `~/.codecatalyst/workflows/`フォルダに保存されます。ソースリポジトリが存在しない場合は、1 つの を作成します。
6. ブランチ フィールドで、ワークフロー定義ファイルが存在するブランチを選択します。
7. [作成] を選択します。

   Amazon はリポジトリとブランチ情報をメモリに CodeCatalyst 保存しますが、ワークフローはまだコミットされていません。
8. ビジュアル を選択します。
9. ワークフローを構築します。

   a. (オプション) ワークフロー図で、ソースボックスとトリガーボックスを選択します。トリガーペインが表示されます。トリガーを追加するには、トリガーの追加を選択します。詳細については、「プッシュ、プル、またはスケジュールトリガーの追加」を参照してください。

   b. + Actions (左上) を選択します。Actions カタログが表示されます。

   c. アクション内のプラス記号 (+) を選択して、ワークフローに追加します。右側のペインを使用して、 アクションを設定します。詳細については、「 CodeCatalyst ワークフローへのアクションの追加」を参照してください。

   d. (オプション) ワークフロープロパティ (右上) を選択します。ワークフロープロパティペインが表示されます。ワークフロー名の実行モードとコンピューティングを設定します。詳細については、「実行のキューイング動作の設定」および「ワークフローのコンピューティング環境とランタイム環境の Docker イメージの設定」を参照してください。
10. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
11. コミット を選択し、コミットワークフローダイアログボックスで次の操作を行います。

    a. ワークフローファイル名 には、デフォルト名のままにするか、独自の名前を入力します。

    b. コミットメッセージ には、デフォルトのメッセージを残すか、独自のメッセージを入力します。

    c. リポジトリとブランチ で、ワークフロー定義ファイルのソースリポジトリとブランチを選択します。これらのフィールドは、ワークフローの作成ダイアログボックスで前に指

定したリポジトリとブランチに設定する必要があります。必要に応じて、リポジトリとブランチを今すぐ変更できます。

> **Note**
>
> ワークフロー定義ファイルをコミットした後は、別のリポジトリやブランチに関連付けることができないため、慎重に選択してください。

d. ワークフロー定義ファイルをコミットするには、コミットを選択します。

YAML

YAML エディタを使用してワークフローを作成するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの作成 を選択します。

   ワークフローの作成ダイアログボックスが表示されます。
5. ソースリポジトリ フィールドで、ワークフロー定義ファイルが存在するソースリポジトリを選択します。ファイルは、選択したリポジトリの `~/.codecatalyst/workflows/` フォルダに保存されます。ソースリポジトリが存在しない場合は、1 つの を作成します。
6. ブランチ フィールドで、ワークフロー定義ファイルが存在するブランチを選択します。
7. [作成] を選択します。

   Amazon はリポジトリとブランチ情報をメモリに CodeCatalyst 保存しますが、ワークフローはまだコミットされていません。
8. YAML を選択します。
9. ワークフローを構築します。

   a. (オプション) YAML コードにトリガーを追加します。詳細については、「プッシュ、プル、またはスケジュールトリガーの追加」を参照してください。

   b. + Actions (左上) を選択します。Actions カタログが表示されます。

c. アクション内のプラス記号 (+) を選択してワークフローに追加します。右側のペインを使用して、 アクションを設定します。詳細については、「 CodeCatalyst ワークフローへのアクションの追加」を参照してください。

d. (オプション) ワークフロープロパティ (右上) を選択します。ワークフロープロパティペインが表示されます。ワークフロー名、実行モード、コンピューティングを設定します。詳細については、「実行のキューイング動作の設定」および「ワークフローのコンピューティング環境とランタイム環境の Docker イメージの設定」を参照してください。

10. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。

11. コミット を選択し、コミットワークフローダイアログボックスで、次の操作を行います。

a. ワークフローファイル名 には、デフォルト名のままにするか、独自の名前を入力します。

b. コミットメッセージ には、デフォルトのメッセージを残すか、独自のメッセージを入力します。

c. リポジトリとブランチ で、ワークフロー定義ファイルのソースリポジトリとブランチを選択します。これらのフィールドは、ワークフローの作成ダイアログボックスで前に指定したリポジトリとブランチに設定する必要があります。必要に応じて、リポジトリとブランチを今すぐ変更できます。

> **Note**
>
> ワークフロー定義ファイルをコミットした後は、別のリポジトリやブランチに関連付けることができないため、慎重に選択してください。

d. ワークフロー定義ファイルをコミットするには、コミットを選択します。

ワークフローの実行

実行はワークフローの 1 回の反復です。実行中、 CodeCatalyst はワークフロー設定ファイルで定義されたアクションを実行し、関連するログ、アーティファクト、変数を出力します。

ワークフロートリガー を使用して、実行を手動で開始することも、自動的に開始することもできます。ワークフロートリガーの例は、ソフトウェア開発者がコミットをメインブランチにプッシュする場合です。

また、誤って開始した場合、処理の途中でワークフローの実行を手動で停止することもできます。

複数のワークフロー実行がほぼ同時に開始される場合は、これらの実行をキューに入れる方法を設定できます。デフォルトのキューイング動作を使用すると、実行が開始された順序で順番にキューに入れられます。または、実行全体を高速化するために、以前のキューから後続の実行を優先 (または「引き継ぐ」) させることができます。ワークフロー実行を並行して実行するように設定して、他の実行を待機しないようにすることもできます。

ワークフロー実行を手動または自動で開始すると、実行のステータスやその他の詳細を表示できます。例えば、いつ開始されたか、誰によって開始されたか、まだ実行されているかどうかを確認できます。

トピック

ワークフローの手動実行の開始

ワークフローの実行を手動で開始するには、次の手順に従います。

> **Note**
>
> トリガー を設定して、ワークフローの実行を自動的に開始することもできます。

ワークフローを手動で開始するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。

4. 実行するワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [実行] を選択します。

トリガーを使用したワークフローの自動実行の開始

ワークフロートリガー 、または単にトリガー を使用すると、コードプッシュなどの特定のイベントが発生したときにワークフローの実行を自動的に開始できます。ソフトウェアデベロッパーが CodeCatalyst コンソールからワークフロー実行を手動で開始する必要がないように、トリガーを設定することもできます。

次の 3 種類のトリガーを使用できます。

- プッシュ — コードプッシュトリガーにより、コミットがプッシュされるたびにワークフロー実行が開始されます。
- プルリクエスト – プルリクエストトリガーは、プルリクエストが作成、改訂、またはクローズされるたびにワークフロー実行を開始します。
- スケジュール – スケジュールトリガーにより、定義したスケジュールでワークフロー実行が開始されます。スケジュールトリガーを使用してソフトウェアの夜間ビルドを実行し、ソフトウェアデベロッパーが翌日に作業できるようにすることを検討してください。

プッシュ、プルリクエスト、スケジュールトリガーは、単独で使用することも、同じワークフロー内で組み合わせて使用することもできます。

> **Tip**
>
> 実行中のトリガーを確認するには、ブループリントを使用してプロジェクトを起動します。ほとんどのブループリントには、トリガー付きのワークフローが含まれています。設計図のワークフロー定義ファイルで `Trigger`プロパティを探します。設計図の詳細については、「設計図を使用したプロジェクトの作成」を参照してください。

トピック

- 一般的なトリガー設定
- 分岐時のトリガーに関する考慮事項

- プッシュ、プル、またはスケジュールトリガーの追加
- トリガーの例

一般的なトリガー設定

このセクションでは、一般的なソフトウェアリリースと分岐戦略のトリガーを設定する方法について説明します。

ソフトウェアリリースと分岐戦略：

- ソースリポジトリにアプリケーションコードがあります。
- main ブランチには、常にリリース可能な最終コードが含まれています。
- ソフトウェアデベロッパーは、mainブランチから機能ブランチに変更を加えます。
- ソフトウェアデベロッパーは、機能の準備ができたら、機能ブランチを にマージするように求めるプルリクエストを作成します。 main

  このプルリクエストで、ソフトウェアデベロッパーの機能ブランチのファイルを使用してアプリケーションを構築してテストするワークフローを自動的に開始します。ただし、デプロイはしません。
- ソフトウェアデベロッパーはビルドとテストをチェックして、すべてが正常であるかどうかを確認します。次に、プルリクエストをブランチにマージします。 main

  マージによって、アプリケーションコードを構築してデプロイするワークフローが自動的に開始されるようにします。

提案されたワークフロー/トリガー設定：

前述のソフトウェア分岐戦略を考慮すると、次の 2 つのワークフローを使用できます。

- ワークフロー 1 は、プルリクエストが作成または改訂されたときにアプリケーションを構築してテストします。
- ワークフロー 2 は、プルリクエストがマージされたときにアプリケーションを構築してデプロイします。

ワークフロー 1 は次のようになります。

```
Triggers:
```

```
  - Type: PULLREQUEST
    Branches:
      - main
    Events:
      - OPEN
      - REVISION
Actions:
  BuildAction:
    instructions-for-building-the-app
  TestAction:
    instructions-for-test-the-app
```

前のトリガーコードは、ソフトウェア開発者が機能ブランチをブランチにマージするよう求めるプルリクエストを作成する (または 1 つの を変更する) たびに、ワークフロー実行を自動的に開始しますmain。 CodeCatalyst は、ソースブランチ (デベロッパーの機能ブランチ) のコードを使用してワークフロー実行を開始します。ワークフローはアプリケーションを構築してデプロイします。

ワークフロー 2 は次のようになります。

```
Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  BuildAction:
    instructions-for-building-the-app
  DeployAction:
    instructions-for-deploying-the-app
```

前のトリガーコードでは、 へのマージmainが発生すると、PUSHトリガーがアクティブ化されます。 は、mainブランチのコード (プルリクエストのコードを含むようになりました) を使用してワークフロー実行 CodeCatalyst を開始します。ワークフローはアプリケーションを構築してデプロイします。

ワークフロー定義ファイルにトリガーを追加する手順については、「」を参照してくださいプッシュ、プル、またはスケジュールトリガーの追加。

トリガーの例とその他の説明については、「」を参照してくださいトリガーの例。

分岐時のトリガーに関する考慮事項

このセクションでは、ブランチを含むトリガーを設定する際の主な考慮事項について説明します。

- 考慮事項 1: プッシュリクエストトリガーとプルリクエストトリガーの両方について、ブランチを指定する場合は、トリガー設定で送信先 (または「送信先」) ブランチを指定する必要があります。ソース (または「from」) ブランチは指定しないでください。

  次の例では、任意のブランチからプッシュしてワークフローをmainアクティブ化します。

  ```
  Triggers:
    - Type: PUSH
      Branches:
        - main
  ```

  次の例では、任意のブランチから へのプルリクエストがワークフローをmainアクティブ化します。

  ```
  Triggers:
    - Type: PULLREQUEST
      Branches:
        - main
      Events:
        - OPEN
        - REVISION
  ```

- 考慮事項 2: プッシュトリガーの場合、ワークフローがアクティブ化されると、ワークフローはワークフロー定義ファイルと送信先ブランチのソースファイルを使用して実行されます。
- 考慮事項 3: プルリクエストトリガーの場合、ワークフローがアクティブ化された後、ワークフローはソースブランチのワークフロー定義ファイルとソースファイルを使用して実行されます (トリガー設定で宛先ブランチを指定している場合でも) 。
- 考慮事項 4: あるブランチでまったく同じトリガーが別のブランチで実行されない場合があります。

  次のプッシュトリガーを検討してください。

  ```
  Triggers:
    - Type: PUSH
      Branches:
        - main
  ```

  このトリガーを含むワークフロー定義ファイルが に存在しmain、 にクローンされている場合test、ワークフローは のファイルを使用して自動的に開始されることはありません test

(ただし、 でファイルを使用する場合は、ワークフローを手動で開始して にすることができますtest) 。考慮事項 1 と 2 を確認して、ワークフローが のファイルを使用して自動的に実行されない理由を理解しますtest。

次のプルリクエストトリガーも考慮してください。

```
Triggers:
  - Type: PULLREQUEST
    Branches:
      - main
    Events:
      - OPEN
      - REVISION
```

このトリガーを含むワークフロー定義ファイルが に存在する場合main、ワークフローは のファイルを使用して実行されませんmain。 (ただし、 からtestブランチを作成するとmain、ワークフローは のファイルを使用して実行されます) test。考慮事項 1 と 3 を確認して、その理由を理解します。

プッシュ、プル、またはスケジュールトリガーの追加

次の手順を使用して、プッシュ、プル、またはスケジュールトリガーをワークフローに追加します。

Visual

トリガーを追加するには (ビジュアルエディタ)

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. ビジュアル を選択します。
7. ワークフロー図で、ソースボックスとトリガーボックスを選択します。
8. 設定ペインで、トリガーの追加 を選択します。

9. トリガーの追加ダイアログボックスで、次のようにフィールドに情報を入力します。

   トリガータイプ

   トリガーのタイプを指定します。次のいずれかの値を使用できます。

   - プッシュ (ビジュアルエディタ) または PUSH (YAML エディタ)

     プッシュトリガーは、変更がソースリポジトリにプッシュされたときにワークフロー実行を開始します。ワークフローの実行では、プッシュ先のブランチ (つまり、送信先ブランチ) 内のファイルを使用します。

   - プルリクエスト (ビジュアルエディタ) または PULLREQUEST (YAML エディタ)

     プルリクエストトリガーは、ソースリポジトリでプルリクエストがオープン、更新、またはクローズされたときにワークフロー実行を開始します。ワークフローの実行では、プル元のブランチ (ソースブランチ) 内のファイルを使用します。

   - スケジュール (ビジュアルエディタ) または SCHEDULE (YAML エディタ)

     スケジュールトリガーは、指定した cron 式で定義されたスケジュールでワークフロー実行を開始します。ブランチの ファイルを使用して、ソースリポジトリ内のブランチごとに個別のワークフロー実行が開始されます。(トリガーがアクティブ化されるブランチを制限するには、ブランチフィールド (ビジュアルエディタ) またはBranchesプロパティ (YAML エディタ) を使用します。)

     スケジュールトリガーを設定するときは、次のガイドラインに従ってください。

     - ワークフローごとに 1 つのスケジュールトリガーのみを使用します。
     - CodeCatalyst スペースに複数のワークフローを定義している場合は、同時に開始するように 10 個までスケジュールすることをお勧めします。
     - トリガーの cron 式は、実行間隔を十分に設定してください。詳細については、「Expression」を参照してください。

   例については、「トリガーの例」を参照してください。

   プルリクエストのイベント

   このフィールドは、プルリクエストトリガータイプを選択した場合にのみ表示されます。

ワークフロー実行を開始するプルリクエストイベントのタイプを指定します。有効な値は次のとおりです。

- プルリクエストが作成される (ビジュアルエディタ) または OPEN (YAML エディタ)

  ワークフローの実行は、プルリクエストの作成時に開始されます。

- プルリクエストがクローズされている (ビジュアルエディタ) または CLOSED (YAML エディタ)

  ワークフロー実行は、プルリクエストがクローズされたときに開始されます。CLOSED イベントの動作はトリッキーで、例から最もよく理解できます。詳細については、「例: プル、ブランチ、および「CLOSED」イベントを含むトリガー」を参照してください。

- プルリクエスト (ビジュアルエディタ) または (YAML エディタ) に新しいリビジョンが加えられました REVISION

  ワークフロー実行は、プルリクエストのリビジョンが作成されると開始されます。最初のリビジョンは、プルリクエストの作成時に作成されます。その後、プルリクエストで指定されたソースブランチに誰かが新しいコミットをプッシュするたびに、新しいリビジョンが作成されます。プルリクエストトリガーREVISIONにイベントを含めると、 が のスーパーセットであるため、OPENイベントREVISIONを省略できますOPEN。

同じプルリクエストトリガーで複数のイベントを指定できます。

例については、「トリガーの例」を参照してください。

スケジュール

このフィールドは、スケジュールトリガータイプを選択した場合にのみ表示されます。

スケジュールされたワークフロー実行をいつ実行するかを記述する cron 式を指定します。

の Cron 式では、次の 6 つのフィールド構文 CodeCatalyst を使用します。各フィールドはスペースで区切られます。

 # 時間 days-of-month # days-of-week#

cron 式の例

分	時間	曜日	月	曜日	年	意味
0	0	?	*	MON-FRI	*	毎週月曜日から金曜日の午前 0 時 (UTC+0) にワークフローを実行します。
0	2	*	*	?	*	ワークフローを毎日午前 2 時 (UTC+0) に実行します。
15	22	*	*	?	*	ワークフローを毎日午後 10:15 (UTC+0) に実行します。

分	時間	曜日	月	曜日	年	意味
0/30	22-2	?	*	土 - 日	*	ワークフローは、土曜日から日曜日の開始日の午後 10 時から翌日の午前 2 時 (UTC+0) の間、30 分ごとに実行されます。
45	13	L	*	?	2023-2027	2023 年から 2027 年までの月の最終日の午後 1 時 45 分 (UTC+0) にワークフローを実行します。

で cron 式を指定するときは CodeCatalyst、次のガイドラインに従ってください。

- SCHEDULE トリガーごとに 1 つの cron 式を指定します。
- YAML 工ディタで cron 式を二重引用符 (") で囲みます。
- 協定世界時 (UTC) で時刻を指定します。その他のタイムゾーンはサポートされていません。

- 実行の合間に 30 分以上を設定します。より高速なケイデンスはサポートされていません。
- *days-of-month* または *days-of-week*フィールドを指定しますが、両方を指定することはできません。いずれかのフィールドに値またはアスタリスク (*) を指定する場合は、もう一方のフィールドで疑問符 (?) を使用する必要があります。アスタリスクは「all」、疑問符は「any」を意味します。

cron 式のその他の例と、?、、 *などのワイルドカードに関する情報についてはL、Amazon EventBridge ユーザーガイドの「Cron 式のリファレンス」を参照してください。EventBridge と の Cron 式はまったく同じように CodeCatalyst 動作します。

スケジュールトリガーの例については、「」を参照してくださいトリガーの例。

ブランチとブランチパターン

(オプション)

ワークフロー実行を開始するタイミングを知るために、トリガーがモニタリングするソースリポジトリ内のブランチを指定します。正規表現パターンを使用してブランチ名を定義できます。例えば、 `main.*`を使用して、 で始まるすべてのブランチを照合しますmain。

指定するブランチは、トリガータイプによって異なります。

- プッシュトリガーでは、 にプッシュするブランチ、つまり送信先ブランチを指定します。一致するブランチ内のファイルを使用して、一致するブランチごとに 1 つのワークフロー実行が開始されます。

  例: `main.*`、`mainline`
- プルリクエストトリガーでは、 にプッシュするブランチ、つまり送信先ブランチを指定します。ワークフロー定義ファイルとソースブランチ内のソースファイル (一致したブランチではない) を使用して、一致したブランチごとに 1 回のワークフロー実行が開始されます。

  例: `main.*`、`mainline`、 `v1\-.*` (で始まるブランチに一致v1-)
- スケジュールトリガーには、スケジュールされた実行で使用するファイルを含むブランチを指定します。一致するブランチごとに 1 つのワークフロー実行が開始され、一致するブランチのワークフロー定義ファイルとソースファイルが使用されます。

例: `main.*`、`version\-1\.0`

> **Note**
>
> ブランチを指定しない場合、トリガーはソースリポジトリ内のすべてのブランチをモニタリングし、ワークフロー定義ファイルとソースファイルを使用してワークフロー実行を開始します。
>
> - プッシュ先のブランチ (プッシュトリガーの場合) 。詳細については、「例: シンプルなコードプッシュトリガー」を参照してください。
> - プル元のブランチ (プルリクエストトリガーの場合) 。詳細については、「例: シンプルなプルリクエストトリガー」を参照してください。
> - すべてのブランチ (スケジュールトリガー用) 。ソースリポジトリのブランチごとに 1 回のワークフロー実行が開始されます。詳細については、「例: シンプルなスケジュールトリガー」を参照してください。

ブランチとトリガーの詳細については、「」を参照してください分岐時のトリガーに関する考慮事項。

その他の例については、「トリガーの例」を参照してください。

ファイルが変更されました

このフィールドは、プッシュまたはプルリクエストのトリガータイプを選択した場合にのみ表示されます。

ワークフロー実行を開始するタイミングを知るために、トリガーがモニタリングするソースリポジトリ内のファイルまたはフォルダを指定します。正規表現を使用して、ファイル名またはパスを一致させることができます。

例については、「トリガーの例」を参照してください。

10. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
11. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

YAML

トリガーを追加するには (YAML エディタ)

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. YAML を選択します。
7. 次の例をガイドとして使用して、Triggersセクションと基盤となるプロパティを追加します。詳細については、ワークフロー YAML 定義 の「Triggers」を参照してください。

   コードプッシュトリガーは次のようになります。

```
Triggers:
  - Type: PUSH
    Branches:
      - main
```

   プルリクエストトリガーは次のようになります。

```
Triggers:
  - Type: PULLREQUEST
    Branches:
      - main.*
    Events:
      - OPEN
      - REVISION
      - CLOSED
```

   スケジュールトリガーは次のようになります。

```
Triggers:
  - Type: SCHEDULE
    Branches:
      - main.*
```

```
    # Run the workflow at 1:15 am (UTC+0) every Friday until the end of 2023
    Expression: "15 1 ? * FRI 2022-2023"
```

Expression プロパティで使用できる cron 式のその他の例については、「」を参照してくださいExpression。

プッシュ、プルリクエスト、スケジュールトリガーのその他の例については、「」を参照してくださいトリガーの例。

8. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
9. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

トリガーの例

次の例は、ワークフロー定義ファイルにさまざまなタイプのトリガーを追加する方法を示しています。

トピック

- 例: シンプルなコードプッシュトリガー
- 例: シンプルな「メインへのプッシュ」トリガー
- 例: シンプルなプルリクエストトリガー
- 例: シンプルなスケジュールトリガー
- 例: スケジュールとブランチを含むトリガー
- 例: スケジュール、プッシュ、ブランチを含むトリガー
- 例: プルとブランチを含むトリガー
- 例: プル、ブランチ、および「CLOSED」イベントを含むトリガー
- 例: プッシュ、ブランチ、ファイルを含むトリガー

例: シンプルなコードプッシュトリガー

次の例は、ソースリポジトリ内のブランチにコードがプッシュされるたびにワークフロー実行を開始するトリガーを示しています。

このトリガーがアクティブ化されると、 はプッシュ先のブランチ (つまり、送信先ブランチ) 内のファイルを使用してワークフロー実行 CodeCatalyst を開始します。

例えば、コミットを にプッシュするとmain、 は の workfow 定義ファイルとその他のソースファイルを使用してワークフロー実行 CodeCatalyst を開始しますmain。

別の例として、コミットを にプッシュするとfeature-branch-123、 は の workfow 定義ファイルとその他のソースファイルを使用してワークフロー実行 CodeCatalyst を開始しますfeature-branch-123。

```
Triggers:
  - Type: PUSH
```

> **Note**
>
> にプッシュした場合にのみワークフローの実行を開始する場合はmain、「」を参照してください例: シンプルな「メインへのプッシュ」トリガー。

例: シンプルな「メインへのプッシュ」トリガー

次の例は、ソースリポジトリmain内のブランチ、およびmainブランチにのみコードがプッシュされるたびにワークフロー実行を開始するトリガーを示しています。

```
Triggers:
  - Type: PUSH
    Branches:
      - main
```

例: シンプルなプルリクエストトリガー

次の例は、プルリクエストがソースリポジトリで作成または改訂されるたびにワークフロー実行を開始するトリガーを示しています。

このトリガーがアクティブ化されると、 は、ワークフロー定義ファイルと、プル元のブランチ (ソースブランチ) 内の他のソースファイルを使用してワークフロー実行 CodeCatalyst を開始します。

例えば、 というソースブランチfeature-123と という宛先ブランチを使用してプルリクエストを作成するとmain、 は の workfow 定義ファイルとその他のソースファイルを使用してワークフロー実行 CodeCatalyst を開始しますfeature-123。

```
Triggers:
  - Type: PULLREQUEST
    Events:
```

```
      - OPEN
      - REVISION
```

例: シンプルなスケジュールトリガー

次の例は、毎週月曜日から金曜日の午前 0 時 (UTC+0) にワークフロー実行を開始するトリガーを示しています。

このトリガーがアクティブ化されると、 は、このトリガーを持つワークフロー定義ファイルを含むソースリポジトリ内のブランチごとに 1 つのワークフロー実行 CodeCatalyst を開始します。

例えば、ソースリポジトリに 3 つのブランチがある場合、、feature-123、、mainrelease-v1、およびこれらの各ブランチに、次のトリガーを持つワークフロー定義ファイルが含まれている場合、 は のファイルを使用して 3 つのワークフロー実行 CodeCatalyst を開始します。1 つは のファイルを使用しmain、もう 1 つは のファイルrelease-v1を使用しますfeature-123。

```
Triggers:
  - Type: SCHEDULE
    Expression: "0 0 ? * MON-FRI *"
```

Expression プロパティで使用できる cron 式のその他の例については、「」を参照してくださいExpression。

例: スケジュールとブランチを含むトリガー

次の例は、毎日午後 6:15 (UTC+0) にワークフロー実行を開始するトリガーを示しています。

このトリガーがアクティブ化されると、 はmainブランチ内のファイルを使用してワークフロー実行 CodeCatalyst を開始し、 で始まるブランチごとに追加の実行を開始しますrelease-。

例えば、ソースリポジトリbugfix-2に main、、release-v1、 bugfix-1という名前のブランチがある場合、 は のファイルを使用して 2 つのワークフロー実行 CodeCatalyst を開始します。1 つは のファイルを使用しmain、もう 1 つは のファイルを使用しますrelease-v1。bugfix-1 ブランチと bugfix-1ブランチのワークフロー実行は開始されません。

```
Triggers:
  - Type: SCHEDULE
    Expression: "15 18 * * ? *"
    Branches:
      - main
      - release\-.*
```

Expression プロパティで使用できる cron 式のその他の例については、「」を参照してくださいExpression。

例: スケジュール、プッシュ、ブランチを含むトリガー

次の例は、毎日午前 0 時 (UTC+0) に、コードがmainブランチにプッシュされるたびにワークフロー実行を開始するトリガーを示しています。

この例では、以下のようになっています。

- ワークフロー実行は毎日午前 0 時に開始されます。ワークフロー実行では、ワークフロー定義ファイルとmainブランチ内の他のソースファイルを使用します。
- ワークフロー実行は、コミットをmainブランチにプッシュするたびに開始されます。ワークフロー実行では、ワークフロー定義ファイルと、送信先ブランチ () 内の他のソースファイルを使用しますmain。

```
Triggers:
  - Type: SCHEDULE
    Expression: "0 0 * * ? *"
    Branches:
      - main
  - Type: PUSH
    Branches:
      - main
```

Expression プロパティで使用できる cron 式のその他の例については、「」を参照してくださいExpression。

例: プルとブランチを含むトリガー

次の例は、誰かが という送信先ブランチでプルリクエストを開くか変更するたびにワークフロー実行を開始するトリガーを示していますmain。Triggers 設定で指定されたブランチは ですがmain、ワークフローの実行では、ソースブランチ (からプルするブランチ) のワークフロー定義ファイルとその他のソースファイルが使用されます。

```
Triggers:
  - Type: PULLREQUEST
    Branches:
      - main
    Events:
```

```
      - OPEN
      - REVISION
```

例: プル、ブランチ、および「CLOSED」イベントを含むトリガー

次の例は、 で始まるブランチでプルリクエストがクローズされるたびにワークフロー実行を開始するトリガーを示していますmain。

この例では、以下のようになっています。

- で始まる送信先ブランチでプルリクエストを閉じるとmain、ワークフロー実行は、ワークフロー定義ファイルと (現在は閉じている) ソースブランチ内の他のソースファイルを使用して自動的に開始されます。
- プルリクエストがマージされた後にブランチを自動的に削除するようにソースリポジトリを設定した場合、これらのブランチは CLOSED状態になることはありません。つまり、マージされたブランチはプルリクエストCLOSEDトリガーをアクティブ化しません。このシナリオでCLOSEDトリガーをアクティブ化する唯一の方法は、プルリクエストをマージせずに閉じることです。

```
Triggers:
  - Type: PULLREQUEST
    Branches:
      - main.*
    Events:
      - CLOSED
```

例: プッシュ、ブランチ、ファイルを含むトリガー

次の例は、mainブランチの filename.txt ファイルまたはsrcディレクトリ内のファイルに変更が加えられるたびにワークフロー実行を開始するトリガーを示しています。

このトリガーがアクティブ化されると、 はワークフロー定義ファイルとmainブランチ内の他のソースファイルを使用してワークフロー実行 CodeCatalyst を開始します。

```
Triggers:
  - Type: PUSH
    Branches:
      - main
    FilesChanged:
      - filename.txt
      - src\/.*
```

ワークフロー実行の停止

次の手順を使用して、進行中のワークフロー実行を停止します。誤って開始された場合は、実行を停止できます。

ワークフローの実行を停止すると、 は進行中のアクションが完了するのを CodeCatalyst 待ってから、 CodeCatalyst コンソールで実行が停止とマークされます。開始する機会がなかったアクションは開始されず、中止された としてマークされます。

> **Note**
>
> 実行がキューに入っている (つまり、進行中のアクションがない) 場合、実行は直ちに停止します。

ワークフロー実行を停止するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフロー で、実行 を選択し、リストから進行中の実行を選択します。
5. [Stop] (停止) を選択します。

ワークフロー実行のゲート設定

ゲートは、特定の条件が満たされない限り、ワークフローの実行が続行されないようにするために使用できるワークフローコンポーネントです。ゲートの例は承認ゲートです。承認ゲートでは、ワークフローの実行を続行する前にコンソール CodeCatalystで承認を送信する必要があります。

ワークフロー内の一連のアクション間にゲートを追加するか、最初のアクションの前にゲートを追加できます (ソースのダウンロードの直後に実行されます) 。最後のアクションの後にゲートを追加する必要がある場合は、追加することもできます。

ゲートタイプ

現在、Amazon は承認ゲートの 1 つのタイプのゲート CodeCatalyst をサポートしています。詳細については、「ワークフロー実行の承認を要求する」を参照してください。

別のアクションと並行して実行するゲートを設定できますか？

いいえ。ゲートはアクションの前後にのみ実行できます。詳細については、「ゲートとアクション間の依存関係の設定」を参照してください。

ゲートを使用してワークフローの実行を開始できないようにできますか？

はい、資格があります。

ワークフロー実行がタスク を実行できないようにすることができます。これは、 の開始を妨げることとは若干異なります。

ワークフローがタスクを実行できないようにするには、ワークフローの最初のアクションの前にゲートを追加します。このシナリオでは、ワークフロー実行が開始されます。つまり、ソースリポジトリファイルをダウンロードしますが、ゲートがロック解除されるまでタスクを実行できなくなります。

> **Note**
>
> 開始してゲートによってブロックされるワークフローは、スペースクォータおよびその他のクォータあたりの同時ワークフロー実行の最大数に引き続きカウントされます。ワークフロークォータを超えないようにするには、ワークフロートリガーを使用して、ゲートを使用する代わりに条件付きでワークフローを開始することを検討してください。また、ゲートの代わりにプルリクエストの承認ルールを使用することも検討してください。クォータ、トリガー、プルリクエストの承認ルールの詳細については、「」、トリガーを使用したワークフローの自動実行の開始「」、およびワークフローのクォータ「」を参照してくださいプルリクエストを承認ルールとマージするための要件の管理。

ゲートの制限事項

ゲートには次の制限があります。

- Gates をコンピューティング共有機能と組み合わせて使用することはできません。この機能の詳細については、「アクション間でのコンピューティングの共有」を参照してください。
- アクショングループ内でゲートを使用することはできません。アクショングループの詳細については、「」を参照してくださいアクションをアクショングループにグループ化する。

トピック

- ワークフローへのゲートの追加

- ゲートとアクション間の依存関係の設定
- ゲートのメジャー、マイナー、またはパッチバージョンの指定

ワークフローへのゲートの追加

以下の手順を使用して、ワークフローにゲートを追加し、設定します。

ゲートを追加して設定するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. ビジュアル を選択します。
7. 左側で、ゲート を選択します。
8. ゲートカタログでゲートを検索し、プラス記号 (+) を選択して、ワークフローにゲートを追加します。
9. ゲートを設定します。Visual を選択してビジュアルエディタを使用するか、YAML を選択して YAML エディタを使用します。詳細な手順については、以下を参照してください。
   - ワークフローへの「承認」ゲートの追加
10. (オプション) 検証 を選択して、YAML コードが有効であることを確認します。
11. コミット を選択して変更をコミットします。

ゲートとアクション間の依存関係の設定

アクション、アクショングループ、またはゲートの前後に実行するゲートを設定できます。例えば、Deployアクションの前に実行する Approvalゲートを設定できます。この場合、DeployアクションはApprovalゲートに依存するとされます。

ゲートとアクション間の依存関係を設定するには、ゲートまたはアクションの Depends on プロパティを設定します。手順については、「アクション間の依存関係の設定」を参照してください。参照される手順はワークフローアクションを参照しますが、ゲートにも等しく適用されます。

ゲートを使用して プロパティに依存 を設定する方法の例については、「」を参照してください例：「承認」ゲートの設定。

ゲートのメジャー、マイナー、またはパッチバージョンの指定

デフォルトでは、ワークフローにゲートを追加すると、 は 形式を使用してワークフロー定義ファイルにフルバージョン CodeCatalyst を追加します。

v*major*.*minor*.*patch*

例:

```
My-Gate:
  Identifier: aws/approval@v1
```

ワークフローで特定のメジャーバージョンまたはマイナーバージョンのゲートを使用するように、バージョンを延長できます。手順については、「アクションのメジャー、マイナー、またはパッチバージョンの指定」を参照してください。参照されるトピックはワークフローアクションを参照しますが、ゲートにも等しく適用されます。

ワークフロー実行の承認を要求する

ワークフロー実行を続行する前に承認を要求するように設定できます。これを実現するには、ワークフローに承認ゲートを追加する必要があります。承認ゲートは、ユーザーまたはユーザーのセットが CodeCatalyst コンソールで 1 つ以上の承認を送信するまでワークフローを続行できないようにします。すべての承認が行われると、ゲートは「ロック解除」になり、ワークフロー実行を再開できます。

ワークフローで承認ゲートを使用して、開発チーム、運用チーム、リーダーシップチームに、より広範な対象者にデプロイされる前に変更を確認する機会を与えます。

承認ゲートのロックを解除するにはどうすればよいですか？

承認ゲートをロック解除するには、次の条件をすべて満たす必要があります。

- 条件 1: 必要な承認数を送信する必要があります。必要な承認数は設定可能で、各ユーザーは 1 つの承認を送信できます。
- 条件 2: ゲートがタイムアウトする前に、すべての承認を送信する必要があります。ゲートはアクティブ化されてから 14 日後にタイムアウトします。この期間は設定できません。

- 条件 3: ワークフロー実行を拒否するべき人はいません。1 回拒否すると、ワークフローの実行が失敗します。
- 条件 4: (置き換えられた実行モードを使用している場合にのみ適用されます) 。実行を後の実行に置き換えないでください。詳細については、「ワークフロー承認は、キューに入れられた実行モード、置き換えられた実行モード、並列実行モードでどのように機能しますか？」を参照してください。

いずれかの条件が満たされない場合、 はワークフロー CodeCatalyst を停止し、実行ステータスを Failed (条件 1 から 3 の場合) または Superseded (条件 4 の場合) に設定します。

「承認」ゲートを使用するタイミング

通常、アプリケーションやその他のリソースを本番サーバーや品質標準を検証する必要がある環境にデプロイするワークフローでは、承認ゲートを使用します。本番環境へのデプロイの前にゲートを配置することで、レビューワーに新しいソフトウェアリビジョンが一般に公開される前に検証する機会を与えることができます。

誰が承認を提供できますか？

プロジェクトのメンバーであり、寄稿者またはプロジェクト管理者ロールを持つユーザーは、承認を行うことができます。プロジェクトのスペースに属するスペース管理者ロールを持つユーザーは、承認を提供することもできます。

> **Note**
>
> Reviewer ロールを持つユーザーは、承認を提供できません。

承認が必要であることをユーザーに通知するにはどうすればよいですか？

承認が必要であることをユーザーに通知するには、以下を実行する必要があります。

- Slack 通知 CodeCatalyst を送信します。詳細については、「承認通知の設定」を参照してください。
- 承認ボタンと拒否ボタンがある CodeCatalyst コンソールのページに移動し、承認者宛ての E メールまたはメッセージングアプリケーションにそのページの URL を貼り付けます。このページに移動する方法の詳細については、「」を参照してくださいワークフロー実行の承認または拒否。

ワークフロー実行の開始を防ぐために「承認」ゲートを使用できますか？

はい、資格があります。詳細については、「ゲートを使用してワークフローの実行を開始できないようにできますか？」を参照してください。

ワークフロー承認は、キューに入れられた実行モード、置き換えられた実行モード、並列実行モードでどのように機能しますか？

キューに入れられた実行モード、置き換えられた実行モード、または並列実行モードを使用する場合、承認ゲートはアクション と同様に機能します。これらの実行モードに慣れるためにキューに入れられた実行モードについて、、置き換えられた実行モードについて、 並列実行モードについてセクションを読むことをお勧めします。基本的な理解ができたら、このセクションに戻り、承認ゲートが存在するときにこれらの実行モードがどのように機能するかを確認してください。

承認ゲートが存在する場合、実行は次のように処理されます。

- キューに入れられた実行モード を使用している場合、実行は、ゲートで現在承認を待っている実行の背後でキューに入れられます。そのゲートがロック解除される (つまり、すべての承認が付与される) と、キュー内の次の実行はゲートに進み、承認を待ちます。このプロセスは、キューに入れられた実行がゲート を介して処理されるのを続行します one-by-one。 はこのプロセスFigure 1を示しています。
- 置き換えられた実行モード を使用している場合、動作はキューに入れられた実行モードと同じですが、 がゲートでキューに蓄積されるのではなく、新しい実行が以前の実行よりも優先されます (引き継がれます) 。キューはなく、現在ゲートで承認を待っている実行はキャンセルされ、新しい実行に置き換えられます。 はこのプロセスFigure 2を示しています。
- 並列実行モード を使用している場合、実行は並列で開始され、キュー形式はありません。各実行は、その前に実行がないため、すぐにゲートによって処理されます。 はこのプロセスFigure 3を示しています。

図 1:「キューに入れられた実行モード」と承認ゲート

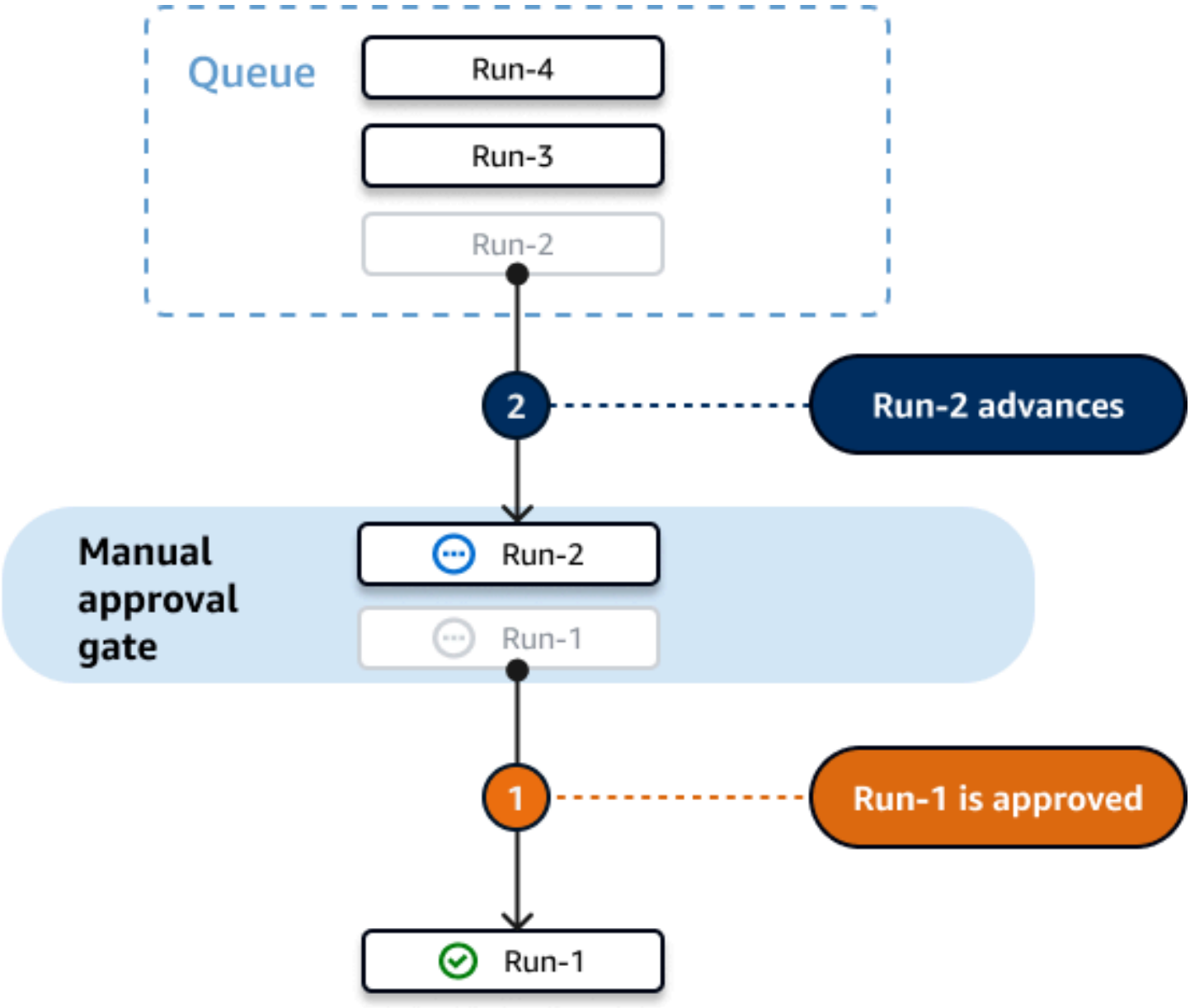


図 2：「スーパーステッド実行モード」と承認ゲート

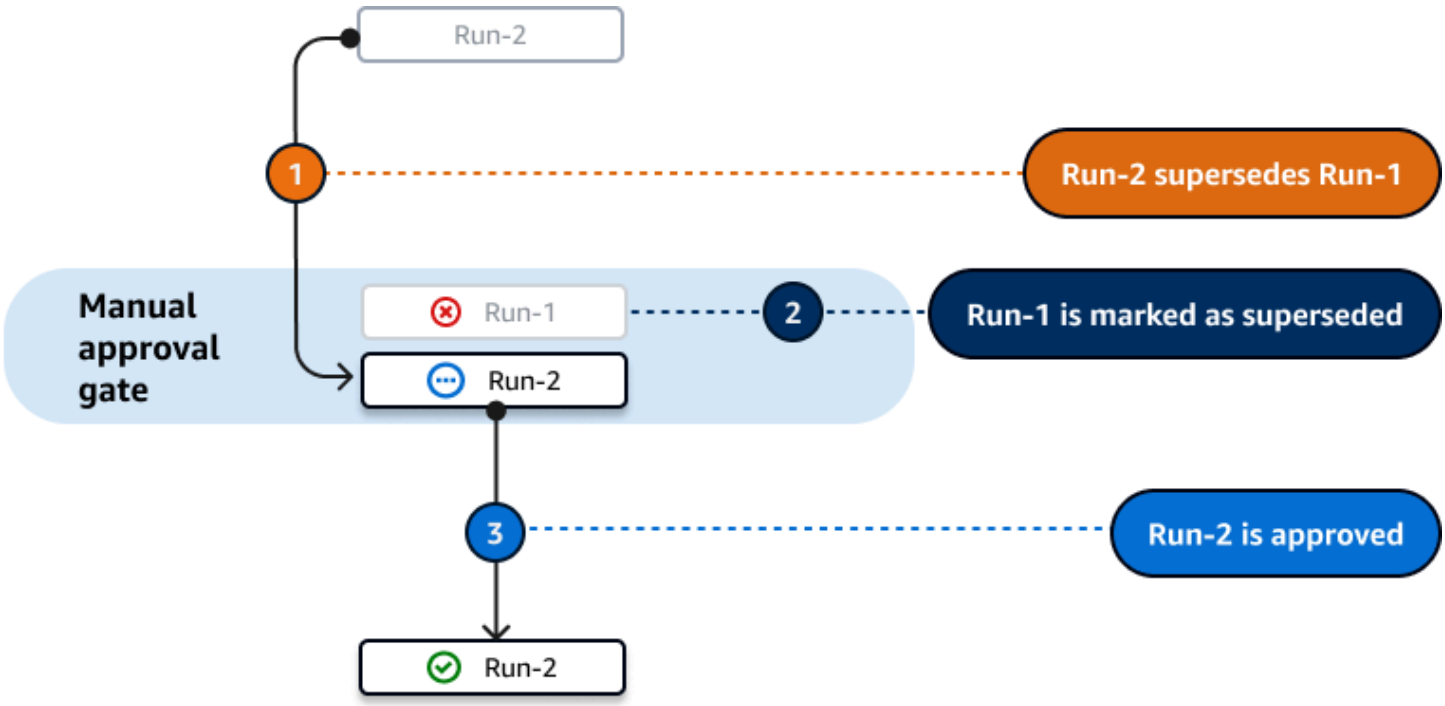
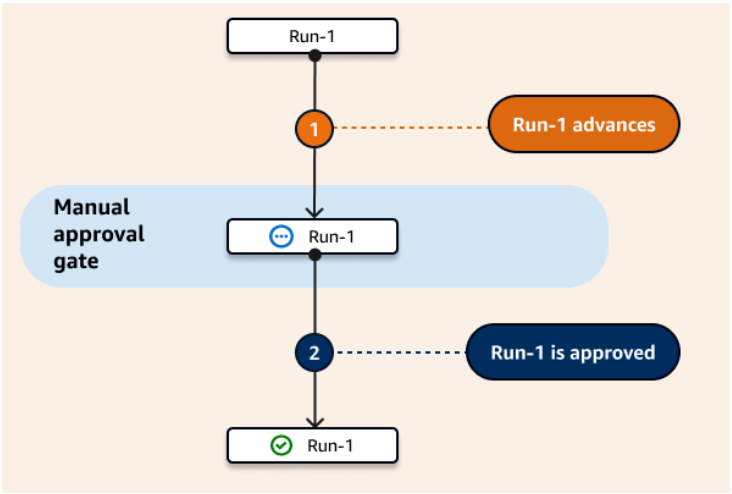
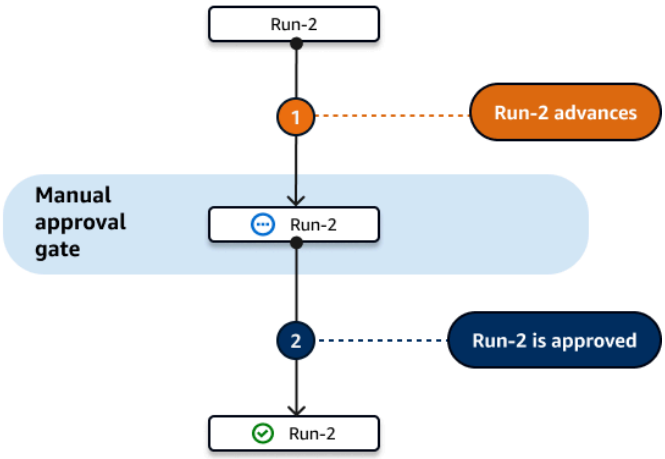


図 3: 「並列実行モード」と承認ゲート





トピック

- 例：「承認」ゲートの設定
- ワークフローへの「承認」ゲートの追加
- 承認通知の設定
- ワークフロー実行の承認または拒否
- 「承認」ゲート YAML 定義

例：「承認」ゲートの設定

次の例は、、、 という 2 つのアクションApproval_01の間に という承認ゲートを追加する方法を示していますStagingProduction。Staging アクションは最初に実行され、Approval_01ゲートは 2 番目、Productionアクションは最後に実行されます。Production アクションは、Approval_01ゲートがロック解除されている場合にのみ実行されます。DependsOn プロパティによりStaging、、Approval_01、および Productionフェーズが順番に実行されます。

承認ゲートの詳細については、「」を参照してくださいワークフロー実行の承認を要求する。

```
Actions:
  Staging: # Deploy to a staging server
    Identifier: aws/ecs-deploy@v1
    Configuration:
    ...
  Approval_01:
    Identifier: aws/approval@v1
    DependsOn:
      - Staging
    Configuration:
      ApprovalsRequired: 2
  Production: # Deploy to a production server
    Identifier: aws/ecs-deploy@v1
    DependsOn:
      - Approval_01
    Configuration:
    ...
```

ワークフローへの「承認」ゲートの追加

承認を要求するようにワークフローを設定するには、ワークフローに承認ゲートを追加する必要があります。以下の手順を使用して、ワークフローに承認ゲートを追加します。

このゲートの詳細については、「」を参照してくださいワークフロー実行の承認を要求する。

Visual

ワークフローに「承認」ゲートを追加するには (ビジュアルエディタ)

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。

3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. 左上で、ゲート を選択します。
7. Gates カタログの承認 で、プラス記号 (+)を選択します。
8. 「入力」を選択し、「依存」フィールドで次の操作を行います。

   このゲートを実行するために正常に実行する必要があるアクション、アクショングループ、またはゲートを指定します。デフォルトでは、ワークフローにゲートを追加すると、そのゲートはワークフローの最後のアクションに応じて設定されます。このプロパティを削除すると、ゲートは何も依存せず、他のアクションよりも先に実行されます。

   > **Note**
   >
   > ゲートは、アクション、アクショングループ、またはゲートの前後に実行するように設定する必要があります。他のアクション、アクショングループ、ゲートと並行して実行するように設定することはできません。

   機能に依存する の詳細については、「」を参照してくださいゲートとアクション間の依存関係の設定。
9. [設定] タブを選択します。
10. ゲート名 フィールドで、次の操作を行います。

    ゲートに付ける名前を指定します。すべてのゲート名はワークフロー内で一意である必要があります。ゲート名は、英数字 (a~z、A~Z、0~9)、ハイフン (-)、アンダースコア (_) に制限されています。スペースは使用できません。引用符を使用してゲート名の特殊文字やスペースを有効にすることはできません。
11. (オプション) 承認数 フィールドで、次の操作を行います。

    承認ゲートのロック解除に必要な承認の最小数を指定します。最小値は 1 です。最大は です2。省略した場合、デフォルトは です1。

> **Note**
>
> ApprovalsRequired プロパティを省略する場合は、ワークフロー定義ファイルからゲートConfigurationのセクションを削除します。

12. (オプション) Validate を選択して、コミットする前にワークフローの YAML コードを検証します。
13. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

YAML

ワークフローに「承認」ゲートを追加するには (YAML エディタ)

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. YAML を選択します。
7. 次の例をガイドとして使用して、 Approvalセクションと基盤となるプロパティを追加します。詳細については、ワークフロー YAML 定義 の「「承認」ゲート YAML 定義」を参照してください。

```
Actions:
  MyApproval_01:
    Identifier: aws/approval@v1
    DependsOn:
      - PreviousAction
    Configuration:
      ApprovalsRequired: 2
```

別の例については、「例 :「承認」ゲートの設定」を参照してください。

8. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。

9. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

承認通知の設定

ワークフローの実行には承認が必要であることをユーザーに通知 CodeCatalyst するために、Slack チャンネルに通知を送信できます。ユーザーは通知を表示し、その中のリンクをクリックします。リンクをクリックすると、ワークフローを承認または拒否できる CodeCatalyst 承認ページが表示されます。

ワークフローが承認、拒否されたこと、または承認リクエストの有効期限が切れたことをユーザーに通知するように通知を設定することもできます。

以下の手順を使用して Slack 通知を設定します。

開始する前に

ワークフローに承認ゲートを追加していることを確認します。詳細については、「ワークフローへの「承認」ゲートの追加」を参照してください。

ワークフロー承認通知を Slack チャネルに送信するには

1. Slack CodeCatalyst で を設定します。詳細については、「Slack 通知を使い始める」を参照してください。
2. 承認が必要なワークフローを含む CodeCatalyst プロジェクトで、まだ有効になっていない場合は通知を有効にします。通知を有効にするには：
   a. プロジェクトに移動し、ナビゲーションペインでプロジェクト設定 を選択します。
   b. 上部で、通知 を選択します。
   c. 通知イベント で、通知の編集 を選択します。
   d. ワークフロー承認保留中を有効にし、 CodeCatalyst が通知を送信する Slack チャネルを選択します。
   e. （オプション) 追加の通知を有効にして、承認、拒否、期限切れの承認についてユーザーに警告します。ワークフロー実行承認済み 、ワークフロー実行拒否 、ワークフロー承認が置き換えられた 、ワークフロー承認がタイムアウトした を有効にできます。各通知の横にある、 が通知を送信する Slack CodeCatalyst チャネルを選択します。
   f. [保存] を選択します。

ワークフロー実行の承認または拒否

承認ゲートを含むワークフロー実行は、承認または拒否する必要があります。ユーザーは、以下から承認または拒否を行うことができます。

- CodeCatalyst コンソール
- チームメンバーから提供されたリンク
- 自動 Slack 通知

ユーザーが承認または拒否すると、この決定を元に戻すことはできません。

> **Note**
>
> ワークフロー実行を承認または拒否できるのは、特定のユーザーのみです。詳細については、「誰が承認を提供できますか？」を参照してください。

開始する前に

ワークフローに承認ゲートを追加していることを確認します。詳細については、「ワークフローへの「承認」ゲートの追加」を参照してください。

CodeCatalyst コンソールからワークフロー実行を承認または拒否するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. ワークフロー図で、承認ゲートを表すボックスを選択します。

   サイドパネルが表示されます。

   > **Note**
   >
   > この時点で、必要に応じてこのページの URL を他の承認者に送信できます。

6. 「決定の確認」で、「 を承認または拒否」を選択します。
7. (オプション) コメント - オプションで、ワークフロー実行を承認または拒否した理由を示すコメントを入力します。
8. [送信] を選択します。

チームメンバーから提供されたリンクからワークフロー実行を承認または拒否するには

1. チームメンバーから送信されたリンクを選択します。(チームメンバーに前の手順を読んでリンクを取得させることができます。)
2. 質問された場合は CodeCatalyst、 にサインインします。

   ワークフロー実行の承認ページにリダイレクトされます。
3. 「決定の確認」で、「 を承認または拒否」を選択します。
4. (オプション) コメント - オプションで、ワークフロー実行を承認または拒否した理由を示すコメントを入力します。
5. [送信] を選択します。

自動 Slack 通知から開始するワークフロー実行を承認または拒否するには

1. Slack 通知が設定されていることを確認します。承認通知の設定 を参照してください。
2. Slack で、承認通知が送信されたチャネルで、承認通知のリンクを選択します。
3. 質問された場合は CodeCatalyst、 にサインインします。

   ワークフロー実行ページにリダイレクトされます。
4. ワークフロー図で、承認ゲートを選択します。
5. 「決定の確認」で、「 を承認または拒否」を選択します。
6. (オプション) コメント - オプションで、ワークフロー実行を承認または拒否した理由を示すコメントを入力します。
7. [送信] を選択します。

「承認」ゲート YAML 定義

以下は、承認ゲートの YAML 定義です。このゲートの使用方法については、「」を参照してくださいワークフロー実行の承認を要求する。

このアクション定義は、より広範なワークフロー定義ファイル内のセクションとして存在します。ファイルの詳細については、「ワークフロー YAML 定義」を参照してください。

> **Note**
>
> 以下の YAML プロパティのほとんどには、対応する UI 要素がビジュアルエディタにあります。UI 要素を検索するには、Ctrl+F を使用します。 要素は、関連付けられた YAML プロパティとともに一覧表示されます。

```
# The workflow definition starts here.
# See ######### for details.

Name: MyWorkflow
SchemaVersion: 1.0
Actions:

# The "Approval" gate definition starts here.
  Approval:
    Identifier: aws/approval@v1
    DependsOn:
      - another-action
    Configuration:
      ApprovalsRequired: number
```

Approval

(必須)

ゲートに付ける名前を指定します。すべてのゲート名はワークフロー内で一意である必要があります。ゲート名は、英数字 (a～z、A～Z、0～9)、ハイフン (-)、アンダースコア (_) に制限されています。スペースは使用できません。引用符を使用してゲート名の特殊文字やスペースを有効にすることはできません。

デフォルト: Approval_nn。

対応する UI: 設定タブ/ゲート名

Identifier

(*Approval*/Identifier)

(必須)

ゲートを識別します。承認ゲートはバージョン をサポートします`1.0.0`。バージョンを短縮しない限り、このプロパティを変更しないでください。詳細については、「アクションのメジャー、マイナー、またはパッチバージョンの指定」を参照してください。

デフォルト: `aws/approval@v1`。

対応する UI: ワークフロー図/Approval_nn/aws/approval@v1 ラベル

DependsOn

(*Approval*/DependsOn)

(オプション)

このゲートを実行するために正常に実行する必要があるアクション、アクショングループ、またはゲートを指定します。デフォルトでは、ワークフローにゲートを追加すると、そのゲートはワークフローの最後のアクションに応じて設定されます。このプロパティを削除すると、ゲートは何も依存せず、他のアクションよりも先に実行されます。

> **Note**
>
> ゲートは、アクション、アクショングループ、またはゲートの前後に実行するように設定する必要があります。他のアクション、アクショングループ、ゲートと並行して実行するように設定することはできません。

機能に依存する の詳細については、「」を参照してくださいゲートとアクション間の依存関係の設定。

対応する UI: の入力タブ/依存

Configuration

(*Approval*/Configuration)

(オプション)

ゲートの設定プロパティを定義できるセクション。

対応する UI: 設定タブ

ApprovalsRequired

(*Approval*/Configuration/ApprovalsRequired)

(オプション)

承認ゲートのロック解除に必要な承認の最小数を指定します。最小値は 1 です。最大は です2。省略した場合、デフォルトは です1。

> **Note**
>
> ApprovalsRequired プロパティを省略する場合は、ワークフロー定義ファイルからゲートConfigurationのセクションを削除します。

対応する UI: 設定タブ/承認数

実行のキューイング動作の設定

デフォルトでは、複数のワークフロー実行が同時に発生すると、 はそれらを CodeCatalyst キューに入れ、開始された順序で 1 つずつ処理します。このデフォルトの動作を変更するには、実行モードを指定します。いくつかの実行モードがあります。

- (デフォルト) キューに入れられた実行モード – CodeCatalyst プロセスは 1 つずつ実行されます
- 置き換えられた実行モード – CodeCatalyst プロセスは 1 つずつ実行され、新しい実行は古い実行を上回ります。
- 並列実行モード – CodeCatalyst プロセスは並列で実行されます

トピック

- キューに入れられた実行モードについて
- 置き換えられた実行モードについて
- 並列実行モードについて
- 実行モードの設定

キューに入れられた実行モードについて

キューに入れられた実行モードでは、実行は順番に行われ、待機中の実行がキューを形成します。

キューはエントリでアクションとアクショングループを指すため、同じワークフロー内に複数のキューを持つことができます (「」を参照Figure 1)。キューに入れられた実行が アクションに入ると、アクションはロックされ、他の実行は入力できません。実行が終了してアクションを終了すると、アクションはロック解除され、次の実行の準備が整います。

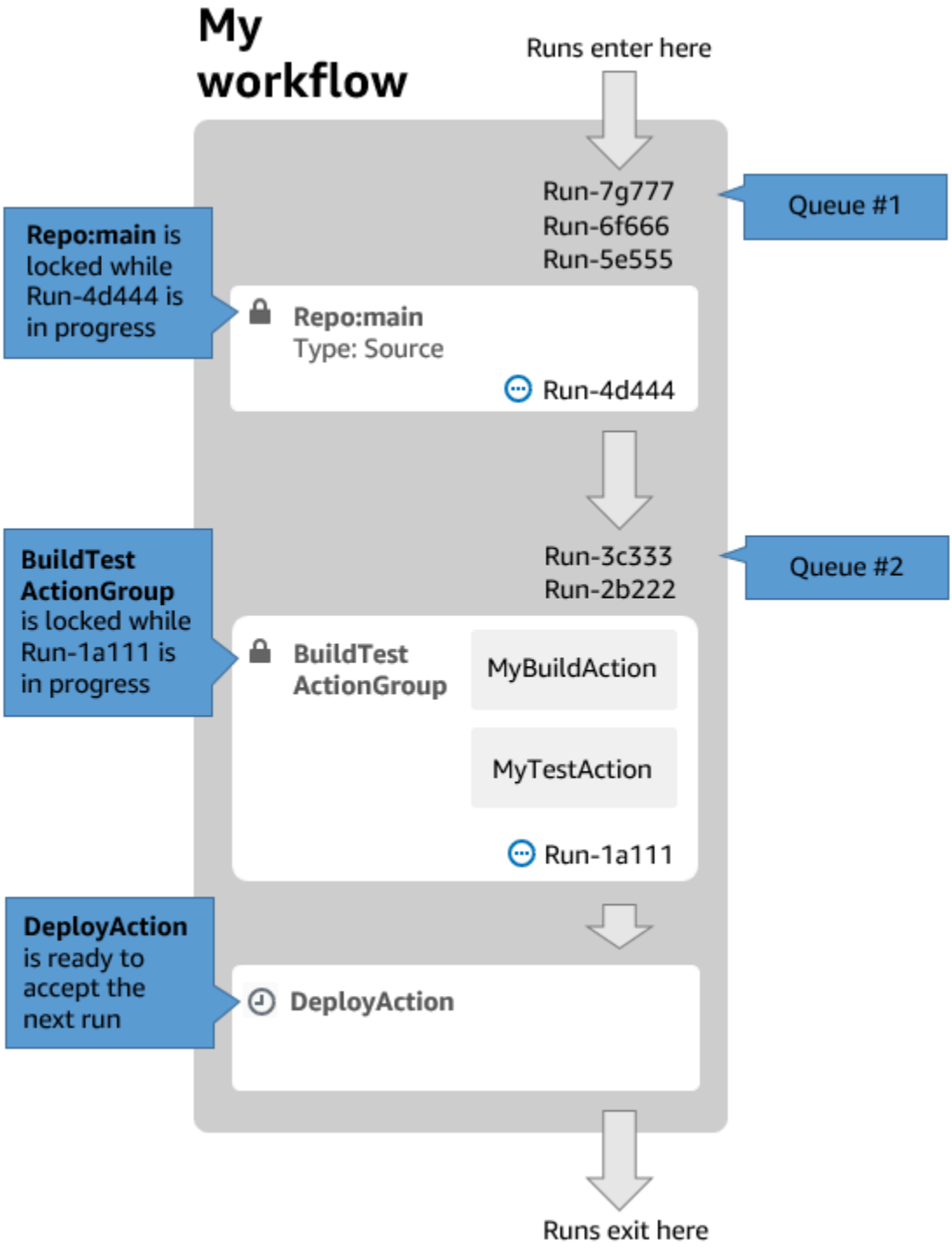
Figure 1 は、キューに入れられた実行モードで設定されたワークフローを示しています。以下が示されています。

- 7 つの実行がワークフローを経由します。
- 2 つのキュー: 1 つは入力ソース (Repo:main) へのエントリの外部、もう 1 つは BuildTestActionGroupアクションへのエントリの外部です。
- 2 つのロックされたブロック: 入力ソース (Repo:main) と BuildTestActionGroup。

ワークフローの実行が処理を終了すると、モノがどのように繰り返されるかは次のとおりです。

- Run-4d444 がソースリポジトリのクローンを作成すると、入力ソースを終了し、Run-3c333 の背後にあるキューに参加します。次に、Run-5e555 が入力ソースに入ります。
- Run-1a111 が構築とテストを完了すると、BuildTestActionGroupアクションを終了し、DeployActionアクションを入力します。次に、Run-2b222 が BuildTestActionGroupアクションに入ります。

図 1:「キューに入れられた実行モード」で設定されたワークフロー



次の場合は、キューに入れられた実行モードを使用します。

- 機能と実行の one-to-one 関係を維持したい場合 - これらの機能は、置き換えモード を使用する場合にグループ化される場合があります。例えば、コミット 1 で機能 1 をマージすると、実行 1 が開始され、コミット 2 で機能 2 をマージすると、実行 2 が開始されます。キューに入れられた

モードの代わりに置き換えられたモードを使用すると、機能 (およびコミット) は実行時にグループ化され、他のモードよりも優先されます。

- 並列モード を使用する場合に発生する可能性のある競合状態や予期しない問題を回避したい。例えば、Wang と Saanvi という 2 人のソフトウェアデベロッパーが、ワークフローをほぼ同時に開始して Amazon ECS クラスターにデプロイする場合、Wang の実行によってクラスターで統合テストが開始され、Saanvi の実行によって新しいアプリケーションコードがクラスターにデプロイされ、Wang のテストが失敗するか、間違ったコードをテストする可能性があります。別の例として、ロックメカニズムを持たないターゲットがあるとします。その場合、2 つの実行が予期しない方法で互いの変更を上書きする可能性があります。
- が実行の処理 CodeCatalyst に使用するコンピューティングリソースの負荷を制限したい。例えば、ワークフローに 3 つのアクションがある場合、同時に最大 3 つの実行を実行できます。一度に実行できる実行数に制限を課すと、実行スループットの予測が容易になります。
- ワークフローによってサードパーティーサービスに対して行われるリクエストの数を制限したい。例えば、ワークフローに Docker Hub からイメージをプルする手順を含むビルドアクションがある場合があります。Docker Hub は、アカウントごとに実行できるプルリクエストの数を 1 時間あたりの特定の数に制限し、制限を超えるとロックアウトされます。キューに入れられた実行モードを使用して実行スループットを遅くすると、1 時間あたりの Docker Hub へのリクエスト数が少なくなり、ロックアウトやビルドおよび実行の失敗の可能性が制限されます。

最大キューサイズ: 50

最大キューサイズに関する注意事項：

- 最大キューサイズは、ワークフロー内のすべてのキューで許可される実行の最大数を指します。
- キューの実行が 50 回を超える場合、 は 51 回目以降の実行 CodeCatalyst を削除します。

障害動作：

アクションの処理中に実行が応答しなくなった場合、その実行の背後にある実行は、アクションがタイムアウトするまでキューに保持されます。アクションは 1 時間後にタイムアウトします。

アクション内で実行が失敗した場合、その背後でキューに入れられた最初の実行は続行できます。

置き換えられた実行モードについて

置き換えられた実行モードは、キューに入れられた実行モードと同じですが、次の点が異なります。

- キューに入れられた実行がキュー内の別の実行に追いついた場合、後の実行は以前の実行よりも優先され (引き継がれます) 、以前の実行はキャンセルされ、「supersed」としてマークされます。
- 最初の箇条書きで説明されている動作の結果として、キューに含めることができる実行は、置き換えられた実行モードが使用されている場合、1 つだけです。

でワークフローをガイドFigure 1として使用し、このワークフローに代替実行モードを適用すると、次のようになります。

- Run-7g777 はキュー内の他の 2 つの実行よりも優先され、キュー #1 に残っている唯一の実行になります。Run-6f666 と Run-5e555 はキャンセルされます。
- Run-3c333 は Run-2b222 よりも優先され、キュー #2 に残っている唯一の実行になります。Run-2b222 はキャンセルされます。

必要に応じて、置き換えられた実行モードを使用します。

- キューに入れられたモードよりもスループットが良い
- キューに入れられたモードよりもサードパーティーサービスへのリクエストがさらに少なくなります。これは、サードパーティーサービスに Docker Hub などのレート制限がある場合に便利です。

並列実行モードについて

並列実行モードでは、実行は互いに独立しており、他の実行が完了するのを待ってから開始しません。キューはなく、実行スループットは、ワークフロー内のアクションが完了するまでにかかる速度によってのみ制限されます。

各ユーザーが独自の機能ブランチを持ち、他のユーザーが共有していないターゲットにデプロイする開発環境では、並列実行モードを使用します。

> **Important**
>
> 本番環境の Lambda 関数など、複数のユーザーがデプロイできる共有ターゲットがある場合は、競合状態が発生する可能性があるため、並列モードを使用しないでください。競合状態は、並列ワークフローの実行が共有リソースを同時に変更しようとして、予期しない結果につながった場合に発生します。

並列実行の最大数: スペースあたり CodeCatalyst 1000

実行モードの設定

実行モードは、キューに入れられた、置き換えられた、または並列に設定できます。デフォルトはキューに入れられます。

実行モードをキューに入れられたか置き換えられたかから並列に変更すると、 はキューに入れられた実行 CodeCatalyst をキャンセルし、アクションによって現在処理されている実行をキャンセルする前に終了することを許可します。

実行モードを並列モードからキューに入れられたモードまたは置き換えられたモードに変更すると、現在実行中 CodeCatalyst のすべての並列実行が完了します。実行モードをキューに入れた、または置き換えた状態に変更した後に開始する実行は、新しいモードを使用します。

Visual

ビジュアルエディタを使用して実行モードを変更するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. 右上で、ワークフロープロパティ を選択します。
7. 詳細 を展開し、実行モード で次のいずれかを選択します。
   a. キューに登録済み – 「」を参照 キューに入れられた実行モードについて
   b. 置き換え – 「」を参照してください。 置き換えられた実行モードについて
   c. 並列 – 「」を参照してください。 並列実行モードについて
8. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
9. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

YAML

YAML エディタを使用して実行モードを変更するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. YAML を選択します。
7. RunMode プロパティを次のように追加します。

```
Name: Workflow_6d39
SchemaVersion: "1.0"
RunMode: QUEUED|SUPERSEDED|PARALLEL
```

詳細については、「」の最上位のプロパティ「」セクションの「 RunModeプロパティの説明」を参照してくださいワークフロー YAML 定義。

8. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
9. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

ワークフロー実行間のファイルのキャッシュ

ファイルキャッシュを有効にすると、ビルドアクションとテストアクションはディスク上のファイルをキャッシュに保存し、後続のワークフロー実行でそのキャッシュから復元します。キャッシュは、実行間で変更されていない依存関係を構築またはダウンロードすることによるレイテンシーを短縮します。 はフォールバックキャッシュ CodeCatalyst もサポートしています。これは、必要な依存関係の一部を含む部分的なキャッシュを復元するために使用できます。これにより、キャッシュミスによるレイテンシーの影響を軽減できます。

> **Note**
>
> ファイルキャッシュは、Amazon の CodeCatalyst ビルドアクションとテストアクションでのみ使用でき、EC2 コンピューティングタイプ を使用するように設定されている場合にのみ使用できます。

トピック

ファイルキャッシュについて

ファイルキャッシュを使用すると、データを複数のキャッシュに整理できます。各キャッシュは FileCachingプロパティで参照されます。各キャッシュは、指定されたパスで指定されたディレクトリを保存します。指定されたディレクトリは、今後のワークフロー実行で復元されます。以下は、cacheKey1および という名前の複数のキャッシュでキャッシュするための YAML スニペットの例ですcacheKey2。

```
Actions:
  BuildMyNpmApp:
    Identifier: aws/build@v1
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      Steps:
        - Run: npm install
        - Run: npm run test
    Caching:
      FileCaching:
        cacheKey1:
          Path: file1.txt
          RestoreKeys:
             - restoreKey1
        cacheKey2:
          Path: /root/repository
          RestoreKeys:
```

```
            - restoreKey2
            - restoreKey3
```

> **Note**
>
> CodeCatalyst は、ローカルキャッシュとリモートキャッシュで構成される多層キャッシュを使用します。プロビジョニングされたフリートまたはオンデマンドマシンがローカルキャッシュでキャッシュミスに遭遇すると、依存関係はリモートキャッシュから復元されます。その結果、一部のアクションの実行では、リモートキャッシュのダウンロードによるレイテンシーが発生する可能性があります。

CodeCatalyst はキャッシュアクセス制限を適用して、あるワークフローのアクションが別のワークフローからキャッシュを変更できないようにします。これにより、ビルドやデプロイに影響する誤ったデータをプッシュする可能性のある他のユーザーから各ワークフローが保護されます。制限は、キャッシュをすべてのワークフローとブランチのペアリングに分離するキャッシュスコープで適用されます。例えば、 ブランチ`workflow-A`のファイルキャッシュ`feature-A`は、 兄弟ブランチ`workflow-A`のファイルキャッシュとは異なります`feature-B`。

キャッシュミスは、ワークフローが指定されたファイルキャッシュを検索して見つけられない場合に発生します。これは、新しいブランチの作成時や、新しいキャッシュが参照され、まだ作成されていない場合など、複数の理由で発生する可能性があります。また、キャッシュの有効期限が切れたときに発生することもあります。デフォルトでは、キャッシュが最後に使用された日から 14 日後に発生します。キャッシュミスを軽減し、キャッシュヒット率を高めるために、 はフォールバックキャッシュ CodeCatalyst をサポートします。フォールバックキャッシュは代替キャッシュであり、キャッシュの古いバージョンである可能性のある部分キャッシュを復元する機会を提供します。キャッシュは、最初に で`FileCaching`プロパティ名に一致するものを検索することで復元され、見つからない場合は が評価されます`RestoreKeys`。プロパティ名とすべての の両方にキャッシュミスがある場合`RestoreKeys`、キャッシュはベストエフォートであり、保証されないため、ワークフローは引き続き実行されます。

キャッシュの作成

以下の手順を使用して、ワークフローにキャッシュを追加できます。

Visual

ビジュアルエディタを使用してキャッシュを追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. ビジュアル を選択します。
7. ワークフロー図で、キャッシュを追加するアクションを選択します。
8. [設定] を選択します。
9. ファイルキャッシュ - オプション で、次のようにキャッシュを追加を選択し、フィールドに情報を入力します。

   キー

   プライマリキャッシュプロパティ名の名前を指定します。キャッシュプロパティ名は、ワークフロー内で一意である必要があります。各アクションには、 に最大 5 つのエントリを含めることができますFileCaching。

   [Path] (パス)

   キャッシュに関連するパスを指定します。

   復元キー - オプション

   プライマリキャッシュプロパティが見つからない場合にフォールバックとして使用する復元キーを指定します。復元キー名はワークフロー内で一意である必要があります。各キャッシュには、 に最大 5 つのエントリを含めることができますRestoreKeys。
10. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
11. コミット を選択し、コミットメッセージを入力し、再度コミットを選択します。

YAML

YAML エディタを使用してキャッシュを追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. YAML を選択します。
7. ワークフローアクションで、次のようなコードを追加します。

```
action-name:
  Configuration:
    Steps: ...
  Caching:
    FileCaching:
      key-name:
        Path: file-path
        # # Specify any additional fallback caches
        # RestoreKeys:
        #   - restore-key
```

8. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
9. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

ファイルキャッシュの制約

プロパティ名 と の制約は次のとおりですRestoreKeys。

- 名前はワークフロー内で一意である必要があります。
- 名前は英数字 (A ~ Z、a ~ z、0 ~ 9)、ハイフン (-)、アンダースコア (_) に制限されています。
- 名前は最大 180 文字です。
- 各アクションには、 に最大 5 つのキャッシュを含めることができますFileCaching。

- 各キャッシュには、 に最大 5 つのエントリを含めることができますRestoreKeys。

パスの制約は次のとおりです。

- アスタリスク (*) は使用できません。
- パスは最大 255 文字です。

ワークフローの実行ステータスと詳細の表示

1 回のワークフロー実行、または複数の実行のステータスと詳細を同時に表示できます。

可能な実行状態のリストについては、「」を参照してくださいワークフロー実行状態。

> **Note**
>
> ワークフロー実行ステータスとは異なるワークフローステータスを表示することもできます。詳細については、「ワークフローステータスの表示」を参照してください。

トピック

1 回の実行のステータスと詳細の表示

1 回のワークフロー実行のステータスと詳細を表示して、成功したかどうか、完了した時刻、または誰が開始したか、何が開始したかを確認することができます。

1 回の実行のステータスと詳細を表示するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。

4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. ワークフローの名前で、実行 を選択します。
6. 実行履歴 の「実行 ID」列で、実行を選択します。例えば Run-95a4d です。
7. 実行の名前で、次のいずれかを実行します。

   - ワークフロー実行のアクションとそのステータスを示すワークフロー図を視覚化します (「」を参照ワークフロー実行状態)。このビューには、実行中に使用されるソースリポジトリとブランチも表示されます。

     ワークフロー図でアクションを選択すると、実行中のアクションによって生成されたログ、レポート、出力などの詳細が表示されます。表示される情報は、どのアクションタイプが選択されているかによって異なります。ビルドログまたはデプロイログの表示の詳細については、ビルドアクションの結果の表示「」または「」を参照してくださいデプロイログの表示。
   - 実行に使用されたワークフロー定義ファイルを表示する YAML。
   - ワークフロー実行によって生成されたアーティファクトを表示するアーティファクト。アーティファクトの詳細については、「アーティファクトを使用したワークフロー内のアクション間のデータの共有」を参照してください。
   - ワークフロー実行によって生成されたテストレポートやその他のタイプのレポートを表示するレポート。レポートの詳細については、「品質レポートタイプ」を参照してください。
   - ワークフロー実行によって生成された出力変数を表示する変数。変数の詳細については、「」を参照してくださいワークフローでの変数の設定と使用。

> **Note**
>
> 実行の親ワークフローが削除された場合、この事実を示すメッセージが実行の詳細ページの上部に表示されます。

プロジェクト内のすべての実行のステータスと詳細の表示

プロジェクト内のすべてのワークフロー実行のステータスと詳細を表示して、プロジェクトで行われているワークフローアクティビティの量を把握し、ワークフローの全体的な状態について確認する場合があります。

プロジェクト内のすべての実行のステータスと詳細を表示するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフロー で、実行 を選択します。

   プロジェクト内のすべてのリポジトリのすべてのブランチで、すべてのワークフローのすべての実行が表示されます。

   このページには、次の列が含まれます。

   - 実行 ID – 実行の一意の識別子。実行 ID リンクを選択すると、実行に関する詳細情報が表示されます。
   - ステータス — ワークフロー実行の処理ステータス。実行状態の詳細については、「」を参照してくださいワークフロー実行状態。
   - トリガー – ワークフロー実行を開始した人、コミット、プルリクエスト (PR)、またはスケジュール。詳細については、「トリガーを使用したワークフローの自動実行の開始」を参照してください。
   - ワークフロー – 実行が開始されたワークフローの名前、およびワークフロー定義ファイルが存在するソースリポジトリとブランチ。この情報を表示するには、列の幅を拡大する必要がある場合があります。

     > **Note**
     >
     > この列が 使用不可 に設定されている場合、通常、関連付けられたワークフローが削除または移動されたことが原因です。

   - 開始時刻 – ワークフロー実行が開始された時刻。
   - 期間 — ワークフローの実行の処理にかかった時間。非常に長い期間または非常に短い期間は、問題を示している可能性があります。
   - 終了時刻 – ワークフローの実行が終了した時刻。

特定のワークフローのすべての実行のステータスと詳細の表示

特定のワークフローに関連付けられているすべての実行のステータスと詳細を表示して、ワークフロー内にボトルネックが発生している実行があるかどうかを確認したり、現在進行中または完了した実行を確認したりできます。

特定のワークフローのすべての実行のステータスと詳細を表示するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. ワークフローの名前で、実行 を選択します。

   選択したワークフローに関連付けられた実行が表示されます。

   このページは 2 つのセクションに分かれています。

   - アクティブな実行 – 進行中の実行を表示します。これらの実行は、進行中 のいずれかの状態になります。
   - 実行履歴 – 完了した (つまり、進行中ではない) 実行を表示します。

     実行状態の詳細については、「」を参照してくださいワークフロー実行状態。

ワークフロー図でのワークフローの実行の表示

ワークフローを一緒に進めると、ワークフローのすべての実行のステータスを表示できます。実行はワークフロー図に表示されます (リストビューでは表示されません)。これにより、どの実行がどのアクションによって処理され、どの実行がキューで待機しているかを視覚的に表現できます。

ワークフローを一緒に進行する複数の実行のステータスを表示するには

> **Note**
>
> この手順は、ワークフローがキューに入れられた実行モードまたは置き換えられた実行モードを使用している場合にのみ適用されます。詳細については、「実行のキューイング動作の設定」を参照してください。

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. 表示する実行を含むワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。

   > **Note**
   >
   > 実行ページではなく、ワークフローページを見ていることを確認してください。

5. 左上の「最新状態」タブを選択します。

   ワークフロー図が表示されます。
6. ワークフロー図を確認します。この図は、ワークフロー内で現在進行中のすべての実行と、完了した最新の実行を示しています。具体的には、次のようになります。

   - ソース の前に上部に表示される実行はキューに入れられ、開始を待っています。
   - アクション間で表示される実行はキューに入れられ、次のアクションによって処理されるのを待機します。
   - アクション内に表示される実行は、1 です。現在、 アクションによって処理されています。2 は アクションによって処理されなくなりました。または、3 は アクションによって処理されませんでした (通常は以前のアクションが失敗したためです) 。

ワークフローが実行するアクションの設定

アクションはワークフローの主要な構成要素であり、ワークフローの実行中に実行する作業またはタスクの論理単位を定義します。通常、ワークフローには、設定方法に応じて順番または並行して実行される複数のアクションが含まれます。

トピック

- アクションタイプ
- CodeCatalyst ワークフローへのアクションの追加
- ワークフローからのアクションの削除
- カスタムアクションの開発

- アクションをアクショングループにグループ化する
- 他のアクションに依存するようにアクションを設定する
- アーティファクトを使用したワークフロー内のアクション間のデータの共有
- アクションのメジャー、マイナー、またはパッチバージョンの指定
- 使用可能なアクションのバージョンの確認
- アクションのソースコードの表示
- ワークフローへの GitHub アクションの統合

アクションタイプ

Amazon CodeCatalyst ワークフロー内では、次のタイプのアクションを使用できます。

アクションタイプ

- CodeCatalyst アクション
- CodeCatalyst ラボアクション
- GitHub アクション
- サードパーティーアクション

CodeCatalyst アクション

CodeCatalyst アクションは、 CodeCatalyst 開発チームが作成、保守、および完全にサポートするアクションです。

アプリケーションの構築、テスト、デプロイ、および 関数の呼び出し AWS Lambda などのその他のタスクを実行するための CodeCatalyst アクションがあります。

以下の CodeCatalyst アクションを使用できます。

- Build

  このアクションはアーティファクトを構築し、Docker コンテナでユニットテストを実行します。詳細については、「ビルドアクションの追加」を参照してください。
- Test

  このアクションは、アプリケーションまたはアーティファクトに対して統合テストとシステムテストを実行します。詳細については、「テストアクションの追加」を参照してください。

- Amazon S3 の発行

  このアクションは、アプリケーションアーティファクトを Amazon S3 バケットにコピーします。詳細については、「ワークフローを使用して Amazon S3 にファイルを発行する」を参照してください。

- AWS CDK bootstrap

  このアクションは、 が CDK アプリケーションをデプロイ AWS CDK するために必要なリソースをプロビジョニングします。詳細については、「ワークフローを使用した AWS CDK アプリケーションのブートストラップ」を参照してください。

- AWS CDK デプロイ

  このアクションは、 AWS Cloud Development Kit (AWS CDK) アプリケーションを合成してデプロイします。詳細については、「ワークフローを使用した AWS Cloud Development Kit (AWS CDK) アプリケーションのデプロイ」を参照してください。

- AWS Lambda 呼び出し

  このアクションは AWS Lambda 関数を呼び出します。詳細については、「ワークフローを使用した AWS Lambda 関数の呼び出し」を参照してください。

- GitHub アクション

  このアクションは、 CodeCatalyst ワークフロー内でCodeCatalystアクションを実行できるようにする GitHub アクションです。詳細については、「ワークフローを使用した AWS Lambda 関数の呼び出し」を参照してください。

- AWS CloudFormation スタックをデプロイする

  このアクションは AWS CloudFormation スタックをデプロイします。詳細については、「ワークフローを使用した AWS CloudFormation スタックのデプロイ」を参照してください。

- Amazon ECS にデプロイする

  このアクションは、Amazon ECS タスク定義を登録し、Amazon ECS サービスにデプロイします。詳細については、「ワークフローを使用した Amazon Elastic Container Service (ECS) へのアプリケーションのデプロイ」を参照してください。

- Kubernetes クラスターにデプロイする

  このアクションは、アプリケーションを Kubernetes クラスターにデプロイします。詳細については、「ワークフローを使用した Amazon Elastic Kubernetes Service へのアプリケーションのデプロイ」を参照してください。

- Amazon ECS タスク定義のレンダリング

  このアクションは、コンテナイメージ URI を Amazon ECS タスク定義 JSON ファイルに挿入し、新しいタスク定義ファイルを作成します。詳細については、「ワークフローを使用した Amazon ECS タスク定義ファイルの変更」を参照してください。

CodeCatalyst アクションのドキュメントは、このガイドと各アクションの readme で入手できます。

使用可能な CodeCatalyst アクション、およびワークフローに追加する方法については、「」を参照してください CodeCatalyst ワークフローへのアクションの追加。

CodeCatalyst ラボアクション

Labs CodeCatalyst アクションは、実験的なアプリケーションの実証場所である Amazon CodeCatalyst Labs の一部であるアクションです。 CodeCatalyst Labs アクションは、 AWS サービスとの統合を紹介するために開発されました。

以下の CodeCatalyst Labs アクションを使用できます。

- AWS Amplify ホスティングにデプロイする

  このアクションは、アプリケーションを Amplify ホスティングにデプロイします。
- にデプロイする AWS App Runner

  このアクションは、ソースイメージリポジトリ内の最新のイメージを App Runner にデプロイします。
- Amazon CloudFront および Amazon S3 にデプロイする

  このアクションは、アプリケーションを CloudFront と Amazon S3 にデプロイします。
- でデプロイする AWS SAM

  このアクションは、 () を使用して AWS Serverless Application Model サーバーレスアプリケーションをデプロイしますAWS SAM。
- Amazon CloudFront キャッシュを無効にする

  このアクションは、特定のパスセットの CloudFront キャッシュを無効にします。
- 送信ウェブフック

このアクションにより、ユーザーは HTTPS リクエストを使用してワークフロー内の任意のウェブサーバーにメッセージを送信できます。

- への発行 AWS CodeArtifact

  このアクションは、パッケージを CodeArtifact リポジトリに発行します。

- Amazon SNS に発行する

  このアクションにより、ユーザーはトピックの作成、トピックへの発行、またはトピックへのサブスクライブによって Amazon SNS と統合できます。

- Amazon ECR にプッシュする

  このアクションは、Docker イメージをビルドして Amazon Elastic Container Registry (Amazon ECR) リポジトリに発行します。

- Amazon CodeGuru Security でスキャンする

  このアクションは、設定されたコードパスの zip アーカイブを作成し、 CodeGuru セキュリティを使用してコードスキャンを実行します。

- Terraform Community Edition

  このアクションは、Terraform Community Edition `plan`および `apply`オペレーションを実行します。

CodeCatalyst Labs アクションのドキュメントは、各アクションの readme で入手できます。

CodeCatalyst Labs アクションをワークフローに追加して readme を表示する方法については、「」を参照してください CodeCatalyst ワークフローへのアクションの追加。

GitHub アクション

GitHub アクションは、 GitHub ワークフローでの使用のために開発されたことを除いて、CodeCatalyst アクション とよく似ています。 GitHub アクションの詳細については、GitHub アクションのドキュメントを参照してください。

CodeCatalyst ワークフローのネイティブ GitHub アクションと一緒に CodeCatalyst アクションを使用できます。

便宜上、 CodeCatalyst コンソールではいくつかの一般的な GitHubアクションにアクセスできます。GitHub Marketplace にリストされている任意の GitHub アクションを使用することもできます (いくつかの制限があります)。

GitHub アクションのドキュメントは、各アクションの readme で入手できます。

詳細については、「ワークフローへの GitHub アクションの統合」を参照してください。

サードパーティーアクション

サードパーティーアクションは、サードパーティーベンダーによって作成され、 CodeCatalyst コンソールで使用できるアクションです。サードパーティーアクションの例には、それぞれ Mend と Sonar によって作成された Mend SCA アクションと SonarCloud Scan アクションが含まれます。

サードパーティーアクションのドキュメントは、各アクションの readme で入手できます。サードパーティーベンダーから追加のドキュメントが提供される場合もあります。

ワークフローへのサードパーティーアクションの追加と readme の表示については、「」を参照してください CodeCatalyst ワークフローへのアクションの追加。

CodeCatalyst ワークフローへのアクションの追加

次の手順を使用して、ワークフローにアクションを追加し、設定します。

アクションを追加および設定するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. 左上で + Actions を選択すると、Actions カタログが表示されます。
7. ドロップダウンリストで、次のいずれかを実行します。

   - Amazon CodeCatalyst を選択してCodeCatalyst、、CodeCatalyst ラボ、またはサードパーティーのアクションを表示します。
     - CodeCatalyst アクションにはラベル別の AWS があります。

- CodeCatalyst ラボアクションには、 CodeCatalyst ラボ別のラベルがあります。
- サードパーティーのアクションには**####**別のラベルがあり、*####*はサードパーティーベンダーの名前です。

- アクション のキュレートされたリストGitHubを表示するには、 を選択します。 GitHub

8. アクションカタログでアクションを検索し、次のいずれかを実行します。

- プラス記号 (+) を選択して、ワークフローにアクションを追加します。
- アクションの名前を選択して readme を表示します。

9. アクションを設定します。Visual を選択してビジュアルエディタを使用するか、YAML を選択して YAML エディタを使用します。詳細な手順については、次のリンクを参照してください。

CodeCatalyst アクション を追加する手順については、以下を参照してください。

- ビルドアクションの追加
- テストアクションの追加
- 「Amazon ECS へのデプロイ」アクションの追加
- 「Kubernetes クラスターへのデプロイ」アクションの追加
- AWS CloudFormation 「スタックのデプロイ」アクションの追加
- AWS CDK 「デプロイ」アクションの追加
- AWS CDK 「ブートストラップ」アクションの追加
- Amazon S3 公開」アクションの追加
- AWS Lambda 「呼び出し」アクションの追加
- 「Amazon ECS タスク定義のレンダリング」アクションの追加

CodeCatalyst Labs アクション を追加する手順については、以下を参照してください。

- アクションの readme。readme は、アクションカタログでアクションの名前を選択することで確認できます。

GitHub アクション を追加する手順については、以下を参照してください。

- ワークフローへの GitHub アクションの統合

サードパーティーアクション を追加する手順については、以下を参照してください。

- アクションの readme。readme は、アクションカタログでアクションの名前を選択することで確認できます。

10. （オプション) 検証 を選択して、YAML コードが有効であることを確認します。
11. コミット を選択して変更をコミットします。

ワークフローからのアクションの削除

ワークフローからアクションを削除するには、次の手順に従います。

Visual

ビジュアルエディタを使用してアクションを削除するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. ビジュアル を選択します。
7. ワークフロー図で、削除するアクションで縦の省略記号アイコンを選択し、 の削除を選択します。
8. （オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
9. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

YAML

YAML エディタを使用してアクションを削除するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。

4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. YAML を選択します。
7. 削除するアクションを含む YAML のセクションを見つけます。

   セクションを選択し、キーボードの削除キーを押します。
8. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
9. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

カスタムアクションの開発

Action Development Kit (ADK) を使用して、ワークフローで使用するカスタム CodeCatalyst アクションを開発できます。その後、アクションを CodeCatalyst アクションカタログに公開して、CodeCatalyst 他のユーザーがワークフローでアクションを表示して使用できるようにします。

アクションを開発、テスト、公開するには (高レベルタスク)

1. アクションの開発に必要なツールとパッケージをインストールします。
2. アクションコードを保存する CodeCatalyst リポジトリを作成します。
3. アクションを初期化します。これにより、独自のコードで更新できるアクション定義ファイル (`action.yml`) など、 アクションに必要なソースファイルが配置されます。
4. アクションコードをブートストラップして、アクションプロジェクトを構築、テスト、リリースするために必要なツールとライブラリを取得します。
5. ローカルコンピュータで アクションを構築し、変更をリポジトリにプッシュします CodeCatalyst。
6. ユニットテストを使用してアクションをローカルでテストし、 で ADK 生成ワークフローを実行します CodeCatalyst。
7. CodeCatalyst コンソールの発行ボタンを選択して、アクションを CodeCatalyst アクションカタログに発行します。

詳細な手順については、「Amazon CodeCatalyst Action Development Kit デベロッパーガイド」を参照してください。

アクションをアクショングループにグループ化する

アクショングループには、1 つ以上のアクションが含まれます。アクションをアクショングループにグループ化すると、ワークフローを整理しやすくなり、異なるグループ間の依存関係を設定することもできます。

> **Note**
>
> 他のアクショングループまたはアクション内にアクショングループをネストすることはできません。

トピック

以下の手順に従って、アクショングループを定義します。

Visual

使用できません。YAML を選択して YAML の手順を表示します。

YAML

グループを定義にするには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. YAML を選択します。
7. で`Actions`、次のようなコードを追加します。

```
Actions:
  action-group-name:
    Actions:
      action-1:
        Identifier: aws/build@v1
        Configuration:
          ...
      action-2:
        Identifier: aws/build@v1
        Configuration:
          ...
```

別の例については、「例: 2 つのアクショングループの定義」を参照してください。詳細については、 の の action-group-nameプロパティの説明を参照してくださいアクションワークフロー YAML 定義。

8. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
9. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

例: 2 つのアクショングループの定義

次の例は、 BuildAndTestと の 2 つのアクショングループを定義する方法を示していますDeploy。BuildAndTest グループには 2 つのアクション (Build と Test) が含まれ、Deployグループには 2 つのアクション (DeployCloudFormationStack と) も含まれますDeployToECS。

```
Actions:
  BuildAndTest: # Action group 1
    Actions:
      Build:
        Identifier: aws/build@v1
        Configuration:
          ...
      Test:
        Identifier: aws/managed-test@v1
        Configuration:
  Deploy: #Action group 2
    Actions:
      DeployCloudFormationStack:
        Identifier: aws/cfn-deploy@v1
```

```
      Configuration:
        ...
    DeployToECS:
      Identifier: aws/ecs-deploy@v1
      Configuration:
        ...
```

他のアクションに依存するようにアクションを設定する

デフォルトでは、ワークフローにアクションを追加すると、ビジュアルエディタ に並べて追加されます。つまり、ワークフロー実行を開始すると、アクションは並行して実行されます。アクションを順番に (ビジュアルエディタに垂直に) 実行する場合は、アクション間の依存関係を設定する必要があります。例えば、ビルドTestアクションの後にテストBuildアクションが実行されるように、アクションに依存するアクションを設定できます。

アクションとアクショングループ間の依存関係を設定できます。 one-to-many 依存関係を設定して、1 つのアクションを開始するために他のいくつかのアクションに依存するようにすることもできます。を参照して依存関係を設定するためのガイドライン、依存関係の設定がワークフローの YAML 構文に準拠していることを確認します。

トピック

- アクション間の依存関係の設定
- 依存関係を設定するためのガイドライン
- アクション間の依存関係を設定する方法の例

アクション間の依存関係の設定

ワークフロー内のアクション間の依存関係を設定するには、次の手順を使用します。

Visual

ビジュアルエディタを使用して依存関係を設定するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。

4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. ビジュアル を選択します。
7. ワークフロー図で、別のアクションに依存するアクションを選択します。
8. [入力] タブを選択します。
9. 依存 - オプション で、次の操作を行います。

   このアクションを実行するために正常に実行する必要があるアクション、アクショングループ、またはゲートを指定します。

   「依存」機能の詳細については、「」を参照してください[他のアクションに依存するようにアクションを設定する]。
10. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
11. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

YAML

YAML エディタを使用して依存関係を設定するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. YAML を選択します。
7. 別の に依存するアクションで、次のようなコードを追加します。

```
action-name:
  DependsOn:
    - action-1
```

その他の例については、「アクション間の依存関係を設定する方法の例」を参照してください。一般的なガイドラインについては、「」を参照してください依存関係を設定するためのガイドライン。詳細については、 ワークフロー YAML 定義アクションの の DependsOn プロパティの説明を参照してください。

8. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
9. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

依存関係を設定するためのガイドライン

依存関係を設定するときは、次のガイドラインに従ってください。

- アクションがグループ内にある場合、そのアクションは同じグループ内の他のアクションにのみ依存できます。
- アクションとアクショングループは、YAML 階層の同じレベルの他のアクションとアクショングループに依存しますが、別のレベルでは依存できません。

アクション間の依存関係を設定する方法の例

次の例は、ワークフロー定義ファイルでアクションとグループ間の依存関係を設定する方法を示しています。

トピック

- 例: 単純な依存関係の設定
- 例: アクションに依存するアクショングループの設定
- 例: 別のアクショングループに依存するようにアクショングループを設定する
- 例: 複数のアクションに依存するようにアクショングループを設定する

例: 単純な依存関係の設定

次の例は、 DependsOnプロパティを使用してTestアクションに依存するようにBuildアクションを設定する方法を示しています。

```
Actions:
  Build:
    Identifier: aws/build@v1
```

```
    Configuration:
      ...
  Test:
    DependsOn:
      - Build
    Identifier: aws/managed-test@v1
     Configuration:
       ...
```

例: アクションに依存するアクショングループの設定

次の例は、DeployGroupアクションに依存するようにFirstActionアクショングループを設定する方法を示しています。アクションとアクショングループは同じレベルであることに注意してください。

```
Actions:
  FirstAction: #An action outside an action group
    Identifier: aws/github-actions-runner@v1
    Configuration:
      ...
  DeployGroup: #An action group containing two actions
    DependsOn:
      - FirstAction
    Actions:
      DeployAction1:
      ...
      DeployAction2:
      ...
```

例: 別のアクショングループに依存するようにアクショングループを設定する

次の例は、DeployGroupアクショングループに依存するようにBuildAndTestGroupアクショングループを設定する方法を示しています。アクショングループは同じレベルであることに注意してください。

```
Actions:
  BuildAndTestGroup: # Action group 1
    Actions:
      BuildAction:
      ...
      TestAction:
      ...
```

```
  DeployGroup: #Action group 2
    DependsOn:
      - BuildAndTestGroup
    Actions:
      DeployAction1:
      ...
      DeployAction2:
      ...
```

例: 複数のアクションに依存するようにアクショングループを設定する

次の例は、DeployGroupアクション、FirstActionアクション、SecondActionおよびアクショングループに依存するようにBuildAndTestGroupアクショングループを設定する方法を示しています。DeployGroup は、、FirstAction、SecondActionおよび と同じレベルであることに注意してくださいBuildAndTestGroup。

```
Actions:
  FirstAction: #An action outside an action group
    ...
  SecondAction: #Another action
    ...
  BuildAndTestGroup: #Action group 1
    Actions:
      Build:
      ...
      Test:
      ...
  DeployGroup: #Action group 2
    DependsOn:
      - FirstAction
      - SecondAction
      - BuildAndTestGroup
    Actions:
      DeployAction1:
      ...
      DeployAction2:
      ...
```

アーティファクトを使用したワークフロー内のアクション間のデータの共有

アーティファクトはワークフローアクションの出力であり、通常はファイルのフォルダまたはアーカイブで構成されます。アーティファクトは、ファイルや情報をアクション間で共有できるため、重要です。

例えば、sam-template.ymlファイルを生成するビルドアクションがあっても、デプロイアクションでそれを使用する場合があります。このシナリオでは、アーティファクトを使用して、ビルドアクションがデプロイアクションとsam-template.ymlファイルを共有できるようにします。コードは次のようになります。

```
Actions:
  BuildAction:
    Identifier: aws/build@v1
    Steps:
      - Run: sam package --output-template-file sam-template.yml
    Outputs:
      Artifacts:
        - Name: MYARTIFACT
          Files:
            - sam-template.yml
  DeployAction:
    Identifier: aws/cfn-deploy@v1
    Inputs:
      Artifacts:
        - MYARTIFACT
    Configuration:
      template: sam-template.yml
```

前のコードでは、ビルドアクション (BuildAction) はsam-template.ymlファイルを生成し、それを という出力アーティファクトに追加しますMYARTIFACT。後続のデプロイアクション (DeployAction) は、 を入力MYARTIFACTとして指定し、 sam-template.yml ファイルへのアクセスを許可します。

アーティファクトを出力と入力として指定せずに共有できますか？

はい。アクションの YAML コードの Outputsおよび Inputsセクションで指定しなくても、アクション間でアーティファクトを共有できます。これを行うには、コンピューティング共有を有効にする必要があります。コンピューティング共有の詳細と、アーティファクトがオンになっているときに

アーティファクトを指定する方法については、「」を参照してくださいアクション間でのコンピューティングの共有。

> **Note**
>
> コンピューティング共有機能を使用すると、 `Outputs`および `Inputs`セクションが不要になり、ワークフローの YAML コードを簡素化できますが、この機能には、オンにする前に注意すべき制限があります。これらの制限の詳細については、「」を参照してくださいコンピューティング共有に関する考慮事項。

ワークフロー間でアーティファクトを共有できますか？

いいえ。異なるワークフロー間でアーティファクトを共有することはできませんが、同じワークフロー内のアクション間でアーティファクトを共有することはできます。

トピック

- 出力アーティファクトの定義
- 入力アーティファクトの定義
- アーティファクト内のファイルを参照する
- アーティファクトのダウンロード
- アーティファクトの例

出力アーティファクトの定義

以下の手順に従って、アクションが出力するアーティファクトを定義します。その後、このアーティファクトは他の アクションで使用できるようになります。

> **Note**
>
> すべてのアクションが出力アーティファクトをサポートしているわけではありません。アクションがそれらをサポートしているかどうかを確認するには、以下のビジュアルエディタの手順を実行し、アクションに Outputs タブの Output artifacts ボタンが含まれているかどうかを確認します。「はい」の場合、出力アーティファクトがサポートされます。

Visual

ビジュアルエディタを使用して出力アーティファクトを定義するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. ビジュアル を選択します。
7. ワークフロー図で、アーティファクトを生成するアクションを選択します。
8. [出力] タブを選択します。
9. アーティファクト で、アーティファクトを追加 を選択します。
10. 次のように、アーティファクトを追加 を選択し、フィールドに情報を入力します。

    ビルドアーティファクト名

    アクションによって生成されたアーティファクトの名前を指定します。アーティファクト名はワークフロー内で一意である必要があり、英数字 (a～z、A～Z、0～9) とアンダースコア (_) に制限されます。スペース、ハイフン (-)、その他の特殊文字は使用できません。引用符を使用して、出力アーティファクト名でスペース、ハイフン、その他の特殊文字を有効にすることはできません。

    アーティファクトの例などの詳細については、「」を参照してくださいアーティファクトを使用したワークフロー内のアクション間のデータの共有。

    ビルドによって生成されるファイル

    アクションによって出力されるアーティファクトに CodeCatalyst を含むファイルを指定します。これらのファイルは、実行時にワークフローアクションによって生成され、ソースリポジトリでも使用できます。ファイルパスは、ソースリポジトリまたは前のアクションのアーティファクトに存在し、ソースリポジトリまたはアーティファクトルートを基準にしています。glob パターンを使用してパスを指定できます。例:

- ビルドまたはソースリポジトリの場所のルートにある 1 つのファイルを指定するには、`my-file.jar` を使用します。
- サブディレクトリ内の 1 つのファイルを指定するには、`directory/my-file.jar` または `directory/subdirectory/my-file.jar` を使用します。
- すべてのファイルを指定するには、`"**/*"` を使用します。glob パターン `**` は、任意の数のサブディレクトリにマッチすることを示します。
- `directory` という名前のディレクトリ内のすべてのファイルとディレクトリを指定するには、`"directory/**/*"` を使用します。glob パターン `**` は、任意の数のサブディレクトリにマッチすることを示します。
- `directory` という名前のディレクトリ内のすべてのファイルを指定するが、そのサブディレクトリを含めないようにするには、`"directory/*"` を使用します。

> **Note**
>
> ファイルパスに 1 つ以上のアスタリスク (*) またはその他の特殊文字が含まれている場合は、パスを二重引用符 () で囲みます""。特殊文字の詳細については、「」を参照してください構文のガイドラインと規則。

アーティファクトの例などの詳細については、「」を参照してくださいアーティファクトを使用したワークフロー内のアクション間のデータの共有。

> **Note**
>
> ファイルパスにプレフィックスを追加して、検索するアーティファクトまたはソースを示す必要がある場合があります。詳細については、「ソースリポジトリ内のファイルを参照する」および「アーティファクト内のファイルを参照する」を参照してください。

11. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
12. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

YAML

YAML エディタを使用して出力アーティファクトを定義するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. YAML を選択します。
7. ワークフローアクションで、次のようなコードを追加します。

```
action-name:
  Outputs:
    Artifacts:
      - Name: artifact-name
        Files:
          - file-path-1
          - file-path-2
```

その他の例については、「アーティファクトの例」を参照してください。詳細については、アクションのワークフロー YAML 定義「」を参照してください。

8. (オプション) Validate を選択して、コミットする前にワークフローの YAML コードを検証します。
9. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

入力アーティファクトの定義

別のアクションによって生成されたアーティファクトを使用する場合は、現在のアクションへの入力として指定する必要があります。複数のアーティファクトを入力として指定できる場合があります。これは アクションによって異なります。詳細については、アクションのワークフロー YAML 定義「」を参照してください。

> **Note**
>
> 他のワークフローのアーティファクトを参照することはできません。

別のアクションからのアーティファクトを現在のアクションへの入力として指定するには、次の手順を使用します。

前提条件

開始する前に、他のアクションからアーティファクトを出力していることを確認してください。詳細については、「出力アーティファクトの定義」を参照してください。アーティファクトを出力すると、他の アクションで使用できるようになります。

Visual

アーティファクトをアクションへの入力として指定するには (ビジュアルエディタ)

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. ビジュアル を選択します。
7. ワークフロー図で、アーティファクトを入力として指定するアクションを選択します。
8. 入力 を選択します。
9. アーティファクト - オプション で、次の操作を行います。

   このアクションへの入力として提供する以前のアクションのアーティファクトを指定します。これらのアーティファクトは、以前のアクションで出力アーティファクトとして定義しておく必要があります。

   入力アーティファクトを指定しない場合は、 で少なくとも 1 つのソースリポジトリを指定する必要があります*action-name*/Inputs/Sources。

アーティファクトの例などの詳細については、「」を参照してくださいアーティファクトを使用したワークフロー内のアクション間のデータの共有。

> **Note**
>
> Artifacts - オプションのドロップダウンリストが使用できない場合 (ビジュアルエディタ)、または YAML (YAML エディタ) の検証時に でエラーが発生した場合は、アクションが 1 つの入力のみをサポートしているためである可能性があります。この場合、ソース入力を削除してみてください。

10. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
11. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

YAML

アーティファクトをアクションへの入力として指定するには (YAML エディタ)

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. YAML を選択します。
7. アーティファクトを入力として指定するアクションで、次のようなコードを追加します。

```
action-name:
  Inputs:
    Artifacts:
      - artifact-name
```

その他の例については、「アーティファクトの例」を参照してください。

8. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。

9. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

アーティファクト内のファイルを参照する

アーティファクト内に存在するファイルがあり、ワークフローアクションのいずれかでこのファイルを参照する必要がある場合は、次の手順を実行します。

> **Note**
>
> 「ソースリポジトリ内のファイルを参照する」も参照してください。

アーティファクト内のファイルを参照するには

- ファイルを参照するアクションで、次のようなコードを追加します。

```
Actions:
  My-action:
    Inputs:
      Sources:
        - WorkflowSource
      Artifacts:
        - artifact-name
    Configuration:
      Steps:
        - run: cd $CATALYST_SOURCE_DIR_artifact-name/build-output && cat file.txt
```

前のコードでは、 アクションはアーティ*#####*アーティファクトの build-output ディレクトリを検索して file.txt ファイルを検索し、表示します。

その他の例については、「アーティファクトの例」を参照してください。

> **Note**
>
> アクションの設定方法によっては、$CATALYST_SOURCE_DIR_*artifact-name*/プレフィックスを省略できる場合があります。詳細については、次のガイダンスを参照してください。

変数を参照する方法に関するガイダンス：

- アクションに の項目が 1 つだけ含まれている場合 Inputs (例えば、入力アーティファクトが 1 つ含まれ、ソースが含まれていない場合) 、プレフィックスを省略し、アーティファクトルートに対するファイルパスのみを指定できます。
- ファイルがプライマリ入力にある場合は、プレフィックスを省略することもできます。プライマリ入力は、 であるかWorkflowSource、 がない場合はリストされた最初の入力アーティファクトですWorkflowSource。
- プレフィックスは、使用しているアクションによって異なる場合があります。詳細については、以下のテーブルをご参照ください。

アクションタイプ	使用するファイルパスプレフィックス	例
ビルドアクション 、テストアクション	$CATALYST_SOURCE_DIR_ *artifact-name* /	$CATALYST_SOURCE_DIR_MyArtifact/folder1/file.txt
その他のすべてのアクション	$CATALYST_SOURCE_DIR_ *artifact-name* / または /artifacts/ *current-action-name* /*artifact-name* / (このパスは $CATALYST_SOURCE_DIR_*artifact-name* / へのシンボリックリンクです)	$CATALYST_SOURCE_DIR_MyArtifact/folder1/file.txt または /artifacts/MyCurrentAction/MyArtifact/folder1/file.txt

アーティファクトのダウンロード

トラブルシューティングの目的で、ワークフローアクションによって生成されたアーティファクトをダウンロードして検査できます。ダウンロードできるアーティファクトには 2 種類あります。

- ソースアーティファクト — 実行の開始時に存在していたソースリポジトリコンテンツのスナップショットを含むアーティファクト。

- ワークフローアーティファクト — ワークフローの設定ファイルの Outputsプロパティで定義されているアーティファクト。

ワークフローによって出力されたアーティファクトをダウンロードするには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. ワークフローの名前AUnder、「 を実行」を選択します。
6. 実行履歴 の「実行 ID」列で、実行を選択します。例えば Run-95a4d です。
7. 実行の名前で、アーティファクト を選択します。
8. アーティファクトの横にある のダウンロード を選択します。アーカイブファイルがダウンロードされます。ファイル名は 7 つのランダムな文字で構成されます。
9. 選択したアーカイブ抽出ユーティリティを使用してアーカイブを抽出します。

アーティファクトの例

次の例は、ワークフロー定義ファイルでアーティファクトを出力、入力、参照する方法を示しています。

トピック

- 例: アーティファクトの出力
- 例: 別のアクションによって生成されたアーティファクトの入力
- 例: 複数のアーティファクト内のファイルを参照する
- 例: 単一のアーティファクトでファイルを参照する
- 例: WorkflowSource が存在する場合にアーティファクト内のファイルを参照する

例: アーティファクトの出力

次の例は、2 つの .jar ファイルを含むアーティファクトを出力する方法を示しています。

```
Actions:
```

```
  Build:
    Identifier: aws/build@v1
    Outputs:
      Artifacts:
        - Name: ARTIFACT1
          Files:
            - build-output/file1.jar
            - build-output/file2.jar
```

例: 別のアクションによって生成されたアーティファクトの入力

次の例は、 ARTIFACT4で というアーティファクトを出力しBuildActionA、それを に入力する方法を示していますBuildActionB。

```
Actions:
  BuildActionA:
    Identifier: aws/build@v1
    Outputs:
      Artifacts:
        - Name: ARTIFACT4
          Files:
            - build-output/file1.jar
            - build-output/file2.jar
  BuildActionB:
    Identifier: aws/build@v1
    Inputs:
      Artifacts:
        - ARTIFACT4
    Configuration:
```

例: 複数のアーティファクト内のファイルを参照する

次の例は、 ART6で ART5と という名前の 2 つのアーティファクトを出力しBuildActionC、file5.txt (BuildActionDの下の) で (アーティファクト のART5) と file6.txt (アーティファクト のART6) という名前の 2 つのファイルを参照する方法を示していますSteps。

> **Note**
>
> ファイル参照の詳細については、「」を参照してくださいアーティファクト内のファイルを参照する。

> **Note**
>
> この例は使用されている$CATALYST_SOURCE_DIR_ART5プレフィックスを示していますが、省略できます。これは、 ART5がプライマリ入力 であるためです。プライマリ入力の詳細については、「」を参照してくださいアーティファクト内のファイルを参照する。

```
Actions:
  BuildActionC:
    Identifier: aws/build@v1
    Outputs:
      Artifacts:
        - Name: ART5
          Files:
            - build-output/file5.txt
        - Name: ART6
          Files:
            - build-output/file6.txt
  BuildActionD:
    Identifier: aws/build@v1
    Inputs:
      Artifacts:
        - ART5
        - ART6
    Configuration:
      Steps:
        - run: cd $CATALYST_SOURCE_DIR_ART5/build-output && cat file5.txt
        - run: cd $CATALYST_SOURCE_DIR_ART6/build-output && cat file6.txt
```

例: 単一のアーティファクトでファイルを参照する

次の例は、 ART7で という名前のアーティファクトを 1 つ出力しBuildActionE、 file7.txt (の下ART7) で BuildActionF (アーティファクト) を参照する方法を示していますSteps。

リファレンスで、 のようにbuild-outputディレクトリの前に$CATALYST_SOURCE_DIR_######プレフィックスが必要ないことに注目してください例: 複数のアーティファクト内のファイルを参照する。これは、 で指定された項目が 1 つだけであるためですInputs。

> **Note**
>
> ファイル参照の詳細については、「」を参照してくださいアーティファクト内のファイルを参照する。

```
Actions:
  BuildActionE:
    Identifier: aws/build@v1
    Outputs:
      Artifacts:
        - Name: ART7
          Files:
            - build-output/file7.txt
  BuildActionF:
    Identifier: aws/build@v1
    Inputs:
      Artifacts:
        - ART7
    Configuration:
      Steps:
        - run: cd build-output && cat file7.txt
```

例: WorkflowSource が存在する場合にアーティファクト内のファイルを参照する

次の例は、 ART8で という名前の 1 つのアーティファクトを出力しBuildActionG、 file8.txt (の下にあるART8) で BuildActionH (アーティファクト の) を参照する方法を示していますSteps。

リファレンスが のようにアー$CATALYST_SOURCE_DIR_#######プレフィックスをどのように必要とするかに注目してください例: 複数のアーティファクト内のファイルを参照する。これは、Inputs (ソースとアーティファクト) に複数の項目が指定されているので、ファイルを検索する場所を示すプレフィックスが必要です。

> **Note**
>
> ファイル参照の詳細については、「」を参照してくださいアーティファクト内のファイルを参照する。

```
Actions:
  BuildActionG:
    Identifier: aws/build@v1
    Outputs:
      Artifacts:
        - Name: ART8
          Files:
            - build-output/file8.txt
  BuildActionH:
    Identifier: aws/build@v1
    Inputs:
      Sources:
        - WorkflowSource
      Artifacts:
        - ART8
    Configuration:
      Steps:
        - run: cd $CATALYST_SOURCE_DIR_ART8/build-output && cat file8.txt
```

アクションのメジャー、マイナー、またはパッチバージョンの指定

デフォルトでは、ワークフローにアクションを追加すると、Amazon は次の形式を使用してワークフロー定義ファイルにフルバージョン CodeCatalyst を追加します。

v*major*.*minor*.*patch*

例:

```
My-Build-Action:
  Identifier: aws/build@v1.0.0
```

Identifier プロパティの完全バージョンを短縮して、ワークフローが常に最新のマイナーバージョンまたはパッチバージョンのアクションを使用するようにすることができます。

例えば、次のように指定した場合 :

```
My-CloudFormation-Action:
  Identifier: aws/cfn-deploy@v1.0
```

...最新のパッチバージョンは で1.0.4、アクションは を使用します1.0.4。新しいバージョンがリリースされた場合、例えば の場合1.0.5、アクションは を使用します1.0.5。マイナーバージョンがリリースされた場合、例えば の場合1.1.0、アクションは引き続き を使用します1.0.5。

バージョンを指定する詳細な手順については、以下のトピックのいずれかを参照してください。

以下の手順を使用して、ワークフローで使用するアクションのバージョンを指定します。最新のメジャーバージョンまたはマイナーバージョン、または特定のパッチバージョンを指定できます。

アクションの最新のマイナーバージョンまたはパッチバージョンを使用することをお勧めします。

Visual

使用できません。YAML を選択して YAML の手順を表示します。

YAML

アクションの最新バージョンまたは特定のパッチバージョンを使用するようにワークフローを設定するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. YAML を選択します。
7. バージョンを編集するアクションを見つけます。
8. アクションの `Identifier`プロパティを検索し、バージョンを次のいずれかに設定します。

   - action-identifier@v*major* – ワークフローで特定のメジャーバージョンを使用し、最新のマイナーバージョンとパッチバージョンを自動的に選択できるようにするには、この構文を使用します。
   - action-identifier@*vmajor .minor* – ワークフローで特定のマイナーバージョンを使用し、最新のパッチバージョンを自動的に選択できるようにするには、この構文を使用します。
   - action-identifier@*vmajor .minor .patch* – ワークフローに特定のパッチバージョンを使用させるには、この構文を使用します。

> **Note**
>
> 利用可能なバージョンが不明な場合は、「」を参照してください使用可能なアクションのバージョンの確認。

> **Note**
>
> メジャーバージョンを省略することはできません。

9. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
10. コミット を選択し、コミットメッセージを入力し、再度コミット を選択します。

使用可能なアクションのバージョンの確認

次の手順を使用して、ワークフローで使用できるアクションのバージョンを決定します。

Visual

使用可能なアクションバージョンを確認するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. バージョンを表示するアクションを見つけます。
   a. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
   b. ワークフローの名前を選択するか、作成します。ワークフローの作成については、「」を参照してくださいワークフローの作成。
   c. [編集] を選択します。
   d. 左上で、+ Actions を選択してアクションカタログを開きます。
   e. ドロップダウンリストで、Amazon CodeCatalyst を選択して CodeCatalyst、CodeCatalystラボ、およびサードパーティーのアクションを表示するか、キュレーションされた GitHubアクションを表示するGitHubかを選択します。

f. アクションを検索し、その名前を選択します。プラス記号 (+) を選択しないでください。

アクションの詳細が表示されます。

4. アクションの詳細ダイアログボックスの右上近くで、バージョンドロップダウンリストを選択して、使用可能なアクションのバージョンのリストを表示します。

YAML

使用できません。「ビジュアル」を選択すると、ビジュアルエディタの手順が表示されます。

アクションのソースコードの表示

アクションのソースコードを表示して、リスクのあるコード、セキュリティの脆弱性、またはその他の欠陥が含まれていないことを確認できます。

次の手順を使用して、、CodeCatalyst ラボCodeCatalyst、またはサードパーティーアクションのソースコードを表示します。

> **Note**
>
> GitHub アクション のソースコードを表示するには、 GitHub Marketplace のアクションのページに移動します。このページには、アクションのリポジトリへのリンクがあり、アクションのソースコードを見つけることができます。

> **Note**
>
> ビルド 、テスト 、 CodeCatalyst アクション ???のアクションのソースコードを表示することはできませんGitHub 。

> **Note**
>
> AWS は、Actions またはサードパーティーのアクションの GitHub アクションコードをサポートまたは保証しません。

アクションのソースコードを表示するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. コードを表示するアクションを見つけます。

   a. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
   b. ワークフローの名前を選択するか、作成します。ワークフローの作成については、「」を参照してくださいワークフローの作成。
   c. [編集] を選択します。
   d. 左上で、+ Actions を選択してアクションカタログを開きます。
   e. ドロップダウンリストで Amazon を選択して CodeCatalyst、、 CodeCatalyst Labs、およびサードパーティーのアクション CodeCatalystを表示します。
   f. アクションを検索し、その名前を選択します。プラス記号 (+) を選択しないでください。

   アクションの詳細が表示されます。
4. アクションの詳細ダイアログボックスの下部近くで、ダウンロードを選択します。

   アクションのソースコードが存在する Amazon S3 バケットを示すページが表示されます。Amazon S3 の詳細については、「Amazon Simple Storage Service ユーザーガイドAmazon S3 とは」を参照してください。
5. コードを検査し、品質とセキュリティに対する期待を満たしていることを確認します。

ワークフローへの GitHub アクションの統合

GitHub アクションは、 GitHub ワークフローでの使用のために開発されたことを除いて、CodeCatalyst アクション とよく似ています。 GitHub アクションの詳細については、GitHub アクションのドキュメントを参照してください。

CodeCatalyst ワークフローのネイティブ GitHub アクションと一緒に CodeCatalyst アクションを使用できます。

GitHub アクションを CodeCatalyst ワークフローに追加するには、次の 2 つの方法があります。

- CodeCatalyst コンソールのキュレートされたリストから GitHub アクションを選択できます。いくつかの一般的な GitHub アクションが利用可能です。詳細については、「 GitHub キュレーションされたアクションの追加」を参照してください。

- 使用する GitHub アクションが CodeCatalyst コンソールで利用できない場合は、GitHub アクションアクションを使用して追加できます。

  GitHub Actions アクションは、CodeCatalyst アクションをラップし、 CodeCatalyst ワークフローと互換性を持たせる GitHub アクションです。

  スーパーリンター GitHubアクションをラップするGitHub アクションアクションの例を次に示します。

```
Actions:
  GitHubAction:
    Identifier: aws/github-actions-runner@v1
    Configuration:
      Steps:
        - name: Lint Code Base
          uses: github/super-linter@v4
          env:
            VALIDATE_ALL_CODEBASE: "true"
            DEFAULT_BRANCH: main
```

  前のコードでは、 CodeCatalyst GitHub アクションアクション (で識別`aws/github-actions-runner@v1`) は Super-Linter アクション (で識別`github/super-linter@v4`) をラップし、CodeCatalyst ワークフローで機能します。

  詳細については、「「GitHub アクション」アクションの追加」を参照してください。

前の例に示すように、キュレーションされた GitHub アクションとキュレートされていないアクションの両方をGitHub アクションアクション (`aws/github-actions-runner@v1`) 内にラップする必要があります。アクションが正しく機能するには、ラッパーが必要です。

GitHub アクションと CodeCatalyst アクションの違い

GitHub CodeCatalyst ワークフロー内で使用されるアクションには、アクションと同じレベルのアクセス AWS と および CodeCatalyst 機能 (環境や問題など) との統合はありません CodeCatalyst 。

GitHub アクションはワークフロー内の他の CodeCatalyst アクションとやり取りできますか？

はい。例えば、 GitHub Actions は、他の CodeCatalyst アクションによって生成された変数を入力として使用したり、出力パラメータやアーティファクトを CodeCatalyst アクションと共有したりでき

ます。詳細については、「他のアクションで使用できるように GitHub 出力パラメータをエクスポートする」および「 GitHub 出力パラメータの参照」を参照してください。

どの GitHub アクションを使用できますか？

CodeCatalyst コンソールから利用可能な任意の GitHub アクション、および GitHub Marketplace で利用可能な任意の GitHub アクションを使用できます。Marketplace の GitHub アクションを使用する場合は、次の制限に注意してください。

での GitHub アクションの制限 CodeCatalyst

- GitHub アクションは CodeCatalyst Lambda コンピューティングタイプ では使用できません。

- GitHub アクションは、古いツールを含む 2022 年 11 月のランタイム環境 Docker イメージで実行されます。イメージとツールの詳細については、「」を参照してくださいランタイム環境の Docker イメージの指定。

- GitHub `github`コンテキストに内部的に依存するアクション、または GitHub固有のリソースを参照するアクションは、 では機能しません CodeCatalyst。例えば、次のアクションは では機能しません CodeCatalyst。
  - GitHub リソースの追加、変更、または更新を試みるアクション。例としては、プルリクエストを更新したり、 で問題を作成したりするアクションなどがあります GitHub。
  - にリストされているほぼすべてのアクションhttps://github.com/actions。
- GitHub Docker コンテナアクションであるアクションは機能しますが、デフォルトの Docker ユーザー (ルート) によって実行する必要があります。ユーザー 1001 としてアクションを実行しないでください。 (書き込み時には、ユーザー 1001 は で動作しますが GitHub、 では動作しません) CodeCatalyst。詳細については、「アクション の Dockerfile サポート」の「USER」トピックを参照してください。 GitHub

CodeCatalyst コンソールから利用できる GitHub アクションのリストについては、「」を参照してください GitHub キュレーションされたアクションの追加。

GitHub アクション (高レベルステップ) を追加する方法

CodeCatalyst ワークフローに GitHub アクションを追加する大まかな手順は次のとおりです。

1. CodeCatalyst プロジェクトでは、ワークフロー を作成します。ワークフローでは、アプリケーションを構築、テスト、デプロイする方法を定義します。詳細については、「ワークフローの開始方法」を参照してください。
2. ワークフローで、キュレートされた GitHub アクションを追加するか、 GitHub アクションアクションを追加します。
3. 次のいずれかを実行します。
   - キュレートされたアクションを追加することを選択した場合は、設定します。詳細については、「 GitHub キュレーションされたアクションの追加」を参照してください。
   - 非キュレーションアクションを追加することを選択した場合、GitHubアクションアクション内にアクション GitHub の YAML コード を貼り付けます。このコードは、 GitHub Marketplace の選択した GitHubアクションの詳細ページにあります。コードが で機能するように、コードを少し変更する必要がある場合があります CodeCatalyst。詳細については、「「GitHub アクション」アクションの追加」を参照してください。
4. (オプション) ワークフロー内で、ビルドアクションやテストアクションなどの他のアクションを追加します。詳細については、「でワークフローを使用して構築、テスト、デプロイする CodeCatalyst」を参照してください。
5. ワークフローは、手動またはトリガーを使用して自動的に開始します。ワークフローは、 GitHub アクションとワークフロー内の他のアクションを実行します。詳細については、「ワークフローの手動実行の開始」を参照してください。

詳細な手順については、以下を参照してください。

- GitHub キュレーションされたアクションの追加.
- 「GitHub アクション」アクションの追加.

GitHub アクションは で実行されます GitHubか ?

いいえ。 GitHub アクションは CodeCatalyst、 CodeCatalystのビルドマシン を使用して で実行されます。

GitHub ワークフローを使用することもできますか ?

いいえ。

トピック

- チュートリアル: ワークフローでアクションを使用して GitHubコードをリントする

- 「GitHub アクション」アクションの追加
- GitHub キュレーションされたアクションの追加
- 他のアクションで使用できるように GitHub 出力パラメータをエクスポートする
- GitHub 出力パラメータの参照
- GitHub 「アクション」アクション YAML 定義

チュートリアル: ワークフローでアクションを使用して GitHubコードをリントする

このチュートリアルでは、Amazon CodeCatalyst ワークフローに Super-Linter GitHub アクションを追加します。Super-Linter アクションは、コードを検査し、コードにエラー、フォーマットの問題、疑わしいコンストラクトがある領域を見つけ、結果をコンソールに出力します CodeCatalyst)。ワークフローに linter を追加したら、ワークフローを実行してサンプルの Node.js アプリケーション (`app.js`) をリントします。次に、報告された問題を修正し、ワークフローを再度実行して、修正が機能したかどうかを確認します。

> **Tip**
>
> テンプレート などの AWS CloudFormation YAML ファイルを lint するには、Super-Linter の使用を検討してください。

トピック

- 前提条件
- ステップ 1: ソースリポジトリを作成する
- ステップ 2: app.js ファイルを追加する
- ステップ 3: Super-Linter アクションを実行するワークフローを作成する
- ステップ 4: Super-Linter が検出した問題を修正する
- クリーンアップ

前提条件

開始する前に、以下が必要です。

- が接続されている CodeCatalyst スペース AWS アカウント。詳細については、「スペースの作成」を参照してください。

- CodeCatalyst というスペース内の空のプロジェクトcodecatalyst-linter-project。このプロジェクトを作成するには、最初から開始オプションを選択します。

  詳細については、「Amazon での空のプロジェクトの作成 CodeCatalyst」を参照してください。

ステップ 1: ソースリポジトリを作成する

このステップでは、 でソースリポジトリを作成します CodeCatalyst。このリポジトリを使用して、app.jsこのチュートリアルのサンプルアプリケーションソースファイル を保存します。

ソースリポジトリの詳細については、「」を参照してくださいソースリポジトリの作成。

ソースリポジトリを作成するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクト に移動しますcodecatalyst-linter-project。
3. ナビゲーションペインで [コード] を選択してから、[ソースリポジトリ] を選択します。
4. [リポジトリの追加] を選択し、[リポジトリの作成] を選択します。
5. リポジトリ名 で、次のように入力します。

   ```
   codecatalyst-linter-source-repository
   ```

6. [作成] を選択します。

ステップ 2: app.js ファイルを追加する

このステップでは、ソースリポジトリに app.js ファイルを追加します。には、linter が検出するいくつかのミスがある関数コードapp.jsが含まれています。

app.js ファイルを追加するには

1. CodeCatalyst コンソールで、プロジェクト を選択しますcodecatalyst-linter-project。
2. ナビゲーションペインで [コード] を選択してから、[ソースリポジトリ] を選択します。
3. ソースリポジトリのリストから、リポジトリ を選択しますcodecatalyst-linter-source-repository。
4. ファイル で、ファイルの作成 を選択します。

5. テキストボックスに、次のコードを入力します。

```
// const axios = require('axios')
// const url = 'http://checkip.amazonaws.com/';
let response;
/**
 *
 * Event doc: https://docs.aws.amazon.com/apigateway/latest/developerguide/set-up-
lambda-proxy-integrations.html#api-gateway-simple-proxy-for-lambda-input-format
 * @param {Object} event - API Gateway Lambda Proxy Input Format
 *
 * Context doc: https://docs.aws.amazon.com/lambda/latest/dg/nodejs-prog-model-
context.html
 * @param {Object} context
 *
 * Return doc: https://docs.aws.amazon.com/apigateway/latest/developerguide/set-up-
lambda-proxy-integrations.html
 * @returns {Object} object - API Gateway Lambda Proxy Output Format
 *
 */
exports.lambdaHandler = async (event, context) => {
  try {
    // const ret = await axios(url);
    response = {
      statusCode: 200,
      'body': JSON.stringify({
        message: 'hello world'
        // location: ret.data.trim()
      })
    }
  } catch (err) {
    console.log(err)
    return err
  }

    return response
}
```

6. ファイル名 には、 と入力しますapp.js。その他のデフォルトオプションはそのままにしておきます。

7. [Commit] (コミット) を選択します。

   これで、 という名前のファイルが作成されましたapp.js。

ステップ 3: Super-Linter アクションを実行するワークフローを作成する

このステップでは、ソースリポジトリにコードをプッシュするときに Super-Linter アクションを実行するワークフローを作成します。ワークフローは、YAML ファイルで定義する次の構成要素で構成されます。

- トリガー — このトリガーは、変更をソースリポジトリにプッシュすると、ワークフローの実行を自動的に開始します。トリガーについての詳細は、「トリガーを使用したワークフローの自動実行の開始」を参照してください。
- GitHub 「アクション」アクション – トリガー時に、GitHub アクションアクションは Super-Linter アクションを実行し、これによりソースリポジトリ内のすべてのファイルが検査されます。linter が問題を検出すると、ワークフローアクションは失敗します。

Super-Linter アクションを実行するワークフローを作成するには

1. CodeCatalyst コンソールで、プロジェクト を選択しますcodecatalyst-linter-project。
2. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
3. ワークフローの作成 を選択します。
4. ソースリポジトリ で、 を選択しますcodecatalyst-linter-source-repository。
5. ブランチ で、 を選択しますmain。
6. [作成] を選択します。
7. YAML サンプルコードを削除します。
8. 次の YAML を追加します。

```
Name: codecatalyst-linter-workflow
SchemaVersion: "1.0"
Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  SuperLinterAction:
    Identifier: aws/github-actions-runner@v1
    Configuration:
      Steps:
        github-action-code
```

前述のコードで、この手順の次のステップで指示されているように、 を Super-Linter アクションコード*github-action-code*に置き換えます。

9. GitHub Marketplace の Super-Linter ページに移動します。
10. 下 `steps:` (小文字) でコードを検索し、 `Steps:` (大文字) でワークフローに CodeCatalyst貼り付けます。

    次のコードに示すように、 CodeCatalyst 標準に準拠するように GitHub アクションコードを調整します。

    CodeCatalyst ワークフローは次のようになります。

```
Name: codecatalyst-linter-workflow
SchemaVersion: "1.0"
Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  SuperLinterAction:
    Identifier: aws/github-actions-runner@v1
    Configuration:
      Steps:
        - name: Lint Code Base
          uses: github/super-linter@v4
          env:
            VALIDATE_ALL_CODEBASE: "true"
            DEFAULT_BRANCH: main
```

11. (オプション) 検証 を選択して、コミットする前に YAML コードが有効であることを確認します。
12. コミット を選択し、コミットメッセージ を入力し、`codecatalyst-linter-source-repository`リポジトリ を選択し、再度コミット を選択します。

    これでワークフローが作成されました。ワークフロー実行は、ワークフローの上部で定義されているトリガーが原因で自動的に開始されます。

進行中のワークフロー実行を表示するには

1. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。

2. 先ほど作成したワークフローを選択します: codecatalyst-linter-workflow。
3. ワークフロー図で、 を選択しますSuperLinterAction。
4. アクションが失敗するのを待ちます。この障害は、 linter がコードで問題を検出したために予想されます。
5. CodeCatalyst コンソールを開いたままにして、 に進みますステップ 4: Super-Linter が検出した問題を修正する。

ステップ 4: Super-Linter が検出した問題を修正する

Super-Linter は、ソースリポジトリに含まれているREADME.mdファイルだけでなく、app.jsコードにも問題があるはずです。

linter が検出した問題を修正するには

1. CodeCatalyst コンソールで、ログタブを選択し、リントコードベース を選択します。

   Super-Linter アクションが生成したログが表示されます。
2. Super-Linter ログで、90 行目付近まで下にスクロールします。ここでは、問題の始まりがわかります。これらは次のようになります。

```
/github/workspace/hello-world/app.js:3:13: Extra semicolon.
/github/workspace/hello-world/app.js:9:92: Trailing spaces not allowed.
/github/workspace/hello-world/app.js:21:7: Unnecessarily quoted property 'body'
 found.
/github/workspace/hello-world/app.js:31:1: Expected indentation of 2 spaces but
 found 4.
/github/workspace/hello-world/app.js:32:2: Newline required at end of file but not
 found.
```

3. ソースリポジトリREADME.mdの app.jsと を修正し、変更をコミットします。

> **Tip**
>
> を修正するにはREADME.md、次のように markdownをコードブロックに追加します。
>
> ```
> ```markdown
> Setup examples:
> ...
> ```

```
```

変更により、別のワークフローが自動的に実行されます。ワークフローが終了するのを待ちます。すべての問題を修正した場合、ワークフローは成功します。

クリーンアップ

でクリーンアップして CodeCatalyst 、このチュートリアルのトレースを環境から削除します。

でクリーンアップするには CodeCatalyst

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. を削除しますcodecatalyst-linter-source-repository。
3. を削除しますcodecatalyst-linter-workflow。

このチュートリアルでは、スーパーリンター GitHub アクションを CodeCatalyst ワークフローに追加してコードをリントする方法を学習しました。

「GitHub アクション」アクションの追加

GitHub アクションアクションは、CodeCatalyst GitHub CodeCatalyst アクションをラップしてワークフローに対応させるアクションです。

詳細については、「ワークフローへの GitHub アクションの統合」を参照してください。

GitHub アクションアクションをワークフローに追加するには、次の手順に従います。

> **Tip**
>
> GitHub Actions アクションの使用方法を示すチュートリアルについては、を参照してくださいチュートリアル: ワークフローでアクションを使用して GitHubコードをリントする。

Visual

ビジュアルエディターを使用して「GitHub Actions」アクションを追加するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。

2. プロジェクトを選択します。
3. ナビゲーションペインで [CI/CD] を選択し、[ワークフロー] を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. 「ビジュアル」を選択します。
7. 左上の [+ Actions] を選択してアクションカタログを開きます。
8. ドロップダウンリストから、を選択しますGitHub。
9. GitHub アクションアクションを検索し、次のいずれかを実行します。

   - プラス記号 (+) を選択してアクションをワークフロー図に追加し、設定ウィンドウを開きます。

     または

   - [GitHub アクション] を選択します。[アクションの詳細] ダイアログボックスが表示されます。このダイアログボックスでは:
     - (オプション) アクションのソースコードを表示するには、「ソースを表示」を選択します。
     - 「ワークフローに追加」を選択してアクションをワークフロー図に追加し、設定ペインを開きます。
10. 「入力」タブと「設定」タブで、必要に応じてフィールドを入力します。各フィールドの説明については、を参照してくださいGitHub 「アクション」アクション YAML 定義。このリファレンスでは、YAML エディタとビジュアルエディタの両方に表示される各フィールド (および対応する YAML プロパティ値) に関する詳細情報を提供します。
11. (オプション)「検証」を選択して、コミットする前にワークフローの YAML コードを検証します。
12. [Commit] を選択し、コミットメッセージを入力して、もう一度 [Commit] を選択します。

YAML

YAML エディターを使用して「GitHub アクション」アクションを追加するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. プロジェクトを選択します。

3. ナビゲーションペインで [CI/CD] を選択し、[ワークフロー] を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. YAML を選択します。
7. 左上の [+ Actions] を選択してアクションカタログを開きます。
8. ドロップダウンリストから、を選択しますGitHub。
9. GitHub アクションアクションを検索し、次のいずれかを実行します。

   - プラス記号 (+) を選択してアクションをワークフロー図に追加し、設定ウィンドウを開きます。

     または

   - [GitHub アクション] を選択します。[アクションの詳細] ダイアログボックスが表示されます。このダイアログボックスでは:
     - (オプション) アクションのソースコードを表示するには、「ソースを表示」を選択します。
     - 「ワークフローに追加」を選択してアクションをワークフロー図に追加し、設定ペインを開きます。
10. YAML コードのプロパティを必要に応じて変更します。使用可能な各プロパティの説明は、GitHub 「アクション」アクション YAML 定義に記載されています。
11. (オプション) [検証] を選択して、コミットする前にワークフローの YAML コードを検証します。
12. [Commit] を選択し、コミットメッセージを入力して、もう一度 [Commit] を選択します。

「GitHub アクション」アクション定義

GitHub アクションは、ワークフロー定義ファイル内の YAML プロパティのセットとして定義されます。これらのプロパティの詳細については、GitHub 「アクション」アクション YAML 定義ワークフロー YAML 定義のを参照してください。

GitHub キュレーションされたアクションの追加

GitHub GitHub CodeCatalyst キュレーションされたアクションはコンソールに表示されるアクションで、 GitHub ワークフロー内でアクションを使用する方法の例として役立ちます。 CodeCatalyst

GitHub キュレーションされたアクションは、識別子で識別される CodeCatalyst-authored GitHub Actions アクションにまとめられます。aws/github-actions-runner@v1たとえば、 GitHub キュレーションされたアクション OSS は以下のようになります。TruffleHog

```
Actions:
  TruffleHogOSS_e8:
    Identifier: aws/github-actions-runner@v1
    Inputs:
      Sources:
        - WorkflowSource # This specifies that the action requires this Workflow as a
 source
    Configuration:
      Steps:
        - uses: trufflesecurity/trufflehog@v3.16.0
          with:
            path: ' ' # Required; description: Repository path
            base: ' ' # Required; description: Start scanning from here (usually main
 branch).
            head: ' ' # Optional; description: Scan commits until here (usually dev
 branch).
            extra_args: ' ' # Optional; description: Extra args to be passed to the
 trufflehog cli.
```

前のコードでは、 CodeCatalyst GitHub アクションアクション (で識別aws/github-actions-runner@v1) は TruffleHog OSS アクション (で識別trufflesecurity/trufflehog@v3.16.0) をラップし、ワークフロー内で動作するようにしています。 CodeCatalyst

このアクションを設定するには、with:以下の空の文字列を独自の値に置き換えます。例:

```
Actions:
  TruffleHogOSS_e8:
    Identifier: aws/github-actions-runner@v1
    Inputs:
      Sources:
        - WorkflowSource # This specifies that the action requires this Workflow as a
 source
    Configuration:
      Steps:
        - uses: trufflesecurity/trufflehog@v3.16.0
          with:
            path: ./
```

```
            base: main # Required; description: Start scanning from here (usually main
branch).
            head: HEAD # Optional; description: Scan commits until here (usually dev
branch).
            extra_args: '--debug --only-verified' # Optional; description: Extra args
to be passed to the trufflehog cli.
```

GitHub 精選されたアクションをワークフローに追加するには、以下の手順に従います。 GitHub CodeCatalyst ワークフローでのアクションの使用に関する一般的な情報については、を参照してくださいワークフローへの GitHub アクションの統合。

> **Note**
>
> GitHub キュレーションされたアクションのリストにアクションが表示されない場合でも、アクションアクションを使用してワークフローに追加できます。GitHub 詳細については、「「GitHub アクション」アクションの追加」を参照してください。

Visual

GitHub ビジュアルエディターを使用してキュレーションされたアクションを追加するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで [CI/CD] を選択し、[ワークフロー] を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. 「ビジュアル」を選択します。
7. 左上の [+ Actions] を選択してアクションカタログを開きます。
8. ドロップダウンリストから、を選択しますGitHub。
9. GitHub アクションを参照または検索し、次のいずれかを実行します。

   - プラス記号 (+) を選択してアクションをワークフロー図に追加し、設定ペインを開きます。

     または

- GitHub アクションの名前を選択します。アクションの詳細ダイアログボックスが表示されます。このダイアログボックスでは:
  - (オプション) アクションのソースコードを表示するには、「ソースを表示」を選択します。
  - 「ワークフローに追加」を選択してアクションをワークフロー図に追加し、設定ペインを開きます。

10. 「入力」、「設定」、「出力」の各タブで、必要に応じてフィールドを入力します。各フィールドの説明については、を参照してくださいGitHub 「アクション」アクション YAML 定義。このリファレンスでは、YAML GitHubエディタとビジュアルエディタの両方に表示されるアクションアクションで使用できる各フィールド (および対応する YAML プロパティ値) に関する詳細情報を提供します。

    GitHubキュレーションされたアクションで使用できる設定オプションについては、そのアクションのドキュメントを参照してください。

11. (オプション)「検証」を選択して、コミットする前にワークフローの YAML コードを検証します。
12. [Commit] を選択し、コミットメッセージを入力して、もう一度 [Commit] を選択します。

YAML

YAML GitHub エディターを使用してキュレーションされたアクションを追加するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで [CI/CD] を選択し、[ワークフロー] を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. YAML を選択します。
7. 左上の [+ Actions] を選択してアクションカタログを開きます。
8. ドロップダウンリストから、を選択しますGitHub。
9. GitHub アクションを参照または検索し、次のいずれかを実行します。

- プラス記号 (+) を選択してアクションをワークフロー図に追加し、設定ペインを開きます。

  または

- GitHub アクションの名前を選択します。アクションの詳細ダイアログボックスが表示されます。このダイアログボックスでは:
  - (オプション) アクションのソースコードを表示するには、「ソースを表示」を選択します。
  - 「ワークフローに追加」を選択してアクションをワークフロー図に追加し、設定ペインを開きます。

10. YAML コードのプロパティを必要に応じて変更します。GitHub Actions アクションで使用できる各プロパティの説明は、に記載されています。GitHub 「アクション」アクション YAML 定義

    GitHubキュレーションされたアクションで使用できる設定オプションについては、そのアクションのドキュメントを参照してください。

11. (オプション)「検証」を選択して、コミットする前にワークフローの YAML コードを検証します。
12. [Commit] を選択し、コミットメッセージを入力して、もう一度 [Commit] を選択します。

他のアクションで使用できるように GitHub 出力パラメータをエクスポートする

CodeCatalyst ワークフローでGitHub 出力パラメータを使用できます。

> **Note**
>
> 出力パラメータの別の単語は変数 です。 GitHub はドキュメントで出力パラメータという用語を使用しているため、この用語も使用します。

以下の手順に従って、 GitHub 出力パラメータを GitHub アクションからエクスポートし、他の CodeCatalyst ワークフローアクションで使用できるようにします。

GitHub 出力パラメータをエクスポートするには

1. ワークフローを開き、編集 を選択します。詳細については、「ワークフローの作成」を参照してください。

2. エクスポートする出力パラメータを生成するGitHub アクションアクションで、次のような基本Variablesプロパティを持つ Outputs セクションを追加します。

```
Actions:
  MyGitHubAction:
    Identifier: aws/github-actions-runner@v1
    Outputs:
      Variables:
        - 'step-id_output-name'
```

置換:

- *step-id* と GitHub アクションの stepsセクションの id:プロパティの値。
- *output-name* と GitHub 出力パラメータの名前。

例

次の例は、 という GitHub 出力パラメータをエクスポートする方法を示していますSELECTEDCOLOR。

```
Actions:
  MyGitHubAction:
    Identifier: aws/github-actions-runner@v1
    Outputs:
      Variables:
        - 'random-color-generator_SELECTEDCOLOR'
    Configuration:
      Steps:
        - name: Set selected color
          run: echo "SELECTEDCOLOR=green" >> $GITHUB_OUTPUT
          id: random-color-generator
```

GitHub 出力パラメータの参照

GitHub 出力パラメータを参照するには、以下の手順に従います。

GitHub 出力パラメータを参照するには

1. 「他のアクションで使用できるように GitHub 出力パラメータをエクスポートする」のステップを完了します。

   GitHub 出力パラメータが他のアクションで使用できるようになりました。

2. 出力パラメータVariablesの値に注意してください。これにはアンダースコア (_) が含まれます。

3. 次の構文を使用して出力パラメータを参照してください。

   ```
   ${action-name.output-name}
   ```

   置換:

   - *action-name* と出力パラメータを生成するGitHub アクションの名前 CodeCatalyst (GitHub アクションの nameまたは は使用しないでくださいid) 。
   - *output-name* は、前に書き留めた出力パラメータVariablesの値で指定します。

   例

   ```
   BuildActionB:
     Identifier: aws/build@v1
     Configuration:
       Steps:
         - Run: echo ${MyGitHubAction.random-color-generator_SELECTEDCOLOR}
   ```

   コンテキストを使用した例

   次の例は、 でSELECTEDCOLOR変数を設定して出力しGitHubActionA、 でそれを参照する方法を示していますBuildActionB。

   ```
   Actions:
     GitHubActionA:
       Identifier: aws/github-actions-runner@v1
       Configuration:
         Steps:
           - name: Set selected color
             run: echo "SELECTEDCOLOR=green" >> $GITHUB_OUTPUT
   ```

```
          id: random-color-generator
    Outputs:
      Variables:
      - 'random-color-generator_SELECTEDCOLOR'

  BuildActionB:
    Identifier: aws/build@v1
    Configuration:
      Steps:
        - Run: echo ${GitHubActionA.random-color-generator_SELECTEDCOLOR}
```

GitHub 「アクション」アクション YAML 定義

以下は、GitHubアクションアクションの YAML 定義です。

このアクション定義は、より広範なワークフロー定義ファイル内のセクションとして存在します。ファイルの詳細については、「ワークフロー YAML 定義」を参照してください。

次のコードで YAML プロパティを選択すると、説明が表示されます。

> **Note**
>
> 後続の YAML プロパティのほとんどには、対応する UI 要素がビジュアルエディタにあります。UI 要素を検索するには、Ctrl+F を使用します。 要素は、関連付けられた YAML プロパティとともに一覧表示されます。

```
# The workflow definition starts here.
# See ######### for details.

Name: MyWorkflow
SchemaVersion: 1.0
Actions:

# The action definition starts here.
  action-name:
    Identifier:  aws/github-actions-runner@v1
    DependsOn:
      - dependent-action-name-1
    Compute:
      Fleet: fleet-name
```

```
    Timeout: timeout-minutes
    Environment:
      Name: environment-name
      Connections:
        - Name: account-connection-name
          Role: iam-role-name
    Inputs:
      Sources:
        - source-name-1
        - source-name-2
      Artifacts:
        - artifact-name
      Variables:
        - Name: variable-name-1
          Value: variable-value-1
        - Name: variable-name-2
          Value: variable-value-2
    Outputs:
      Artifacts:
        - Name: output-artifact-1
          Files:
            - github-output/artifact-1.jar
            - "github-output/build*"
        - Name: output-artifact-2
          Files:
            - github-output/artifact-2.1.jar
            - github-output/artifact-2.2.jar
      Variables:
        - variable-name-1
        - variable-name-2
      AutoDiscoverReports:
        Enabled: true | false
        ReportNamePrefix: AutoDiscovered
        IncludePaths:
          - "**/*"
        ExcludePaths:
          - node_modules/cdk/junit.xml
        SuccessCriteria:
          PassRate: percent
          LineCoverage: percent
          BranchCoverage: percent
          Vulnerabilities:
            Severity: CRITICAL|HIGH|MEDIUM|LOW|INFORMATIONAL
            Number: whole-number
```

```
      Reports:
        report-name-1:
          Format: format
          IncludePaths:
            - "*.xml"
          ExcludePaths:
            - report2.xml
            - report3.xml
          SuccessCriteria:
            PassRate: percent
            LineCoverage: percent
            BranchCoverage: percent
            Vulnerabilities:
              Severity: CRITICAL|HIGH|MEDIUM|LOW|INFORMATIONAL
              Number: whole-number
    Configuration
      Steps:
        - github-actions-code
```

action-name

(必須)

アクションの名前を指定します。すべてのアクション名は、ワークフロー内で一意である必要があります。アクション名は、英数字 (a～z、A～Z、0～9)、ハイフン (-)、アンダースコア (_) に制限されています。スペースは使用できません。引用符を使用してアクション名で特殊文字やスペースを有効にすることはできません。

対応する UI: 設定タブ/######

Identifier

(###### /Identifier)

アクションを識別します。バージョンを変更しない限り、このプロパティを変更しないでください。詳細については、「アクションのメジャー、マイナー、またはパッチバージョンの指定」を参照してください。

GitHub アクションアクションaws/github-actions-runner@v1に を使用します。

対応する UI: ワークフロー図/######/aws/github-actions-runner@v1 ラベル

DependsOn

（ ###### /DependsOn ）

(オプション)

このアクションを実行するために正常に実行する必要があるアクション、アクショングループ、またはゲートを指定します。

「依存」機能の詳細については、「」を参照してください他のアクションに依存するようにアクションを設定する。

対応する UI: の入力タブ/依存 - オプション

Compute

（ ###### /Compute ）

(オプション)

ワークフローアクションを実行するために使用されるコンピューティングエンジン。コンピューティングはワークフローレベルまたはアクションレベルで指定できますが、両方を指定することはできません。ワークフローレベルで指定すると、コンピューティング設定はワークフローで定義されたすべてのアクションに適用されます。ワークフローレベルでは、同じインスタンスで複数のアクションを実行することもできます。詳細については、「アクション間でのコンピューティングの共有」を参照してください。

対応する UI: なし

Fleet

（ ###### /Compute/Fleet ）

(オプション)

ワークフローまたはワークフローアクションを実行するマシンまたはフリートを指定します。オンデマンドフリートでは、アクションが開始されると、ワークフローは必要なリソースをプロビジョニングし、アクションが終了するとマシンは破棄されます。オンデマンドフリートの例: `Linux.x86-64.Large`、`Linux.x86-64.XLarge`。オンデマンドフリートの詳細については、「」を参照してくださいオンデマンドフリートのプロパティ。

プロビジョニングされたフリートでは、ワークフローアクションを実行するように一連の専用マシンを設定します。これらのマシンはアイドル状態のままで、すぐにアクションを処理できます。プロビ

ジョニングされたフリートの詳細については、「」を参照してくださいプロビジョニングされたフリートのプロパティ。

Fleet を省略した場合、デフォルトは ですLinux.x86-64.Large。

対応する UI: 設定タブ/コンピューティングフリート - オプション

Timeout

(###### /Timeout)

(オプション)

CodeCatalyst アクションが終了するまでに実行できる時間を分単位で指定します (YAML エディタ)、または時間と分単位で指定します (ビジュアルエディタ)。最小値は 5 分で、最大値は「」で説明されていますワークフローのクォータ。デフォルトのタイムアウトは、最大タイムアウトと同じです。

対応する UI: 設定タブ/タイムアウト - オプション

Environment

(###### /Environment)

(オプション)

アクションで使用する CodeCatalyst 環境を指定します。

環境の詳細については、環境を使用して AWS アカウント および VPCsにデプロイする CodeCatalyst「」および「」を参照してください環境を作成する。

対応する UI: 設定タブ/環境/アカウント/ロール

Name

(###### /Environment/Name)

(Environmentが含まれている場合は必須)

アクションに関連付ける既存の環境の名前を指定します。

対応する UI: 設定タブ/'環境/アカウント/ロール'/環境

Connections

(###### /Environment/Connections)

(Environmentが含まれている場合は必須)

アクションに関連付けるアカウント接続を指定します。で最大 1 つのアカウント接続を指定できますEnvironment。

アカウント接続の詳細については、「」を参照してください接続された AWS リソースへのアクセスを許可する AWS アカウント。アカウント接続を環境に関連付ける方法については、「」を参照してください環境を作成する。

対応する UI: なし

Name

(###### /Environment/Connections/Name)

(オプション)

アカウント接続の名前を指定します。

対応する UI: 設定タブ/'環境/アカウント/ロール'/AWS アカウント接続

Role

(###### /Environment/Connections/Role)

(オプション)

このアクションが Amazon S3 や Amazon ECR などの AWS サービスにアクセスして操作するために使用する IAM ロールの名前を指定します。 Amazon S3 このロールがアカウント接続に追加されていることを確認します。アカウント接続に IAM ロールを追加するには、「」を参照してくださいアカウント接続への IAM ロールの追加。

> **Note**
>
> 十分なアクセス許可があれば、ここでCodeCatalystWorkflowDevelopmentRole-*spaceName*ロールの名前を指定できます。このロールの詳細については、「アカウントとスペースのCodeCatalystWorkflowDevelopmentRole-***spaceName***ロールの作成」を参照してくださ

い。CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールには、セキュリティリスクをもたらす可能性のある非常に広範なアクセス許可があることを理解します。このロールは、セキュリティが懸念されないチュートリアルとシナリオでのみ使用することをお勧めします。

Warning

GitHub アクションアクションに必要なアクセス許可に制限します。より広範なアクセス許可を持つロールを使用すると、セキュリティ上のリスクが生じる可能性があります。

対応する UI: 設定タブ/'環境/アカウント/ロール'/ロール

Inputs

(###### /Inputs)

(オプション)

Inputs セクションでは、ワークフローの実行中にアクションが必要とするデータを定義します。

Note

Actions アクションごとに最大 4 つの入力 (1 つのソースと 3 つのアーティファクト) GitHub が許可されます。変数はこの合計にはカウントされません。

異なる入力 (ソースとアーティファクトなど) にあるファイルを参照する必要がある場合、ソース入力はプライマリ入力、アーティファクトはセカンダリ入力です。セカンダリ入力内のファイルへの参照には、プライマリからファイルを分散するための特別なプレフィックスが付けられます。詳細については、「例: 複数のアーティファクト内のファイルを参照する」を参照してください。

対応する UI: Inputs タブ

Sources

(###### /Inputs/Sources)

(オプション)

アクションに必要なソースリポジトリを表すラベルを指定します。現在、サポートされているラベルは のみです。これはWorkflowSource、ワークフロー定義ファイルが保存されているソースリポジトリを表します。

ソースを省略する場合は、 で少なくとも 1 つの入力アーティファクトを指定する必要がありますaction-name/Inputs/Artifacts。

sources の詳細については、「ワークフローをソースリポジトリに接続する」を参照してください。

対応する UI: 入力タブ/ソース - オプション

Artifacts - input

(###### /Inputs/Artifacts)

(オプション)

このアクションへの入力として提供する以前のアクションのアーティファクトを指定します。これらのアーティファクトは、以前のアクションで出力アーティファクトとして定義しておく必要があります。

入力アーティファクトを指定しない場合は、 で少なくとも 1 つのソースリポジトリを指定する必要がありますaction-name/Inputs/Sources。

アーティファクトの例などの詳細については、「」を参照してくださいアーティファクトを使用したワークフロー内のアクション間のデータの共有。

> **Note**
>
> Artifacts - オプションのドロップダウンリストが使用できない場合 (ビジュアルエディタ)、または YAML (YAML エディタ) の検証時に でエラーが発生した場合は、アクションが 1 つの入力のみをサポートしているためである可能性があります。この場合、ソース入力を削除してみてください。

対応する UI: 入力タブ/アーティファクト - オプション

Variables - input

(###### /Inputs/Variables)

(オプション)

アクションで使用できるようにしたい入力変数を定義する名前と値のペアのシーケンスを指定します。変数名は、英数字 (a～z、A～Z、0～9)、ハイフン (-)、アンダースコア (_) に制限されています。スペースは使用できません。変数名で特殊文字やスペースを有効にするために引用符を使用することはできません。

例を含む変数の詳細については、「」を参照してくださいワークフローでの変数の設定と使用。

対応する UI: 入力タブ/変数 - オプション

Outputs

(###### /Outputs)

(オプション)

ワークフローの実行中に アクションによって出力されるデータを定義します。

対応する UI: 出力タブ

Artifacts - output

(###### /Outputs/Artifacts)

(オプション)

アクションによって生成されたアーティファクトの名前を指定します。アーティファクト名はワークフロー内で一意である必要があり、英数字 (a～z、A～Z、0～9) とアンダースコア (_) に制限されます。スペース、ハイフン (-)、その他の特殊文字は使用できません。引用符を使用して、出力アーティファクト名でスペース、ハイフン、その他の特殊文字を有効にすることはできません。

アーティファクトの詳細については、「」を参照してくださいアーティファクトを使用したワークフロー内のアクション間のデータの共有。

対応する UI: 出力タブ/アーティファクト

Name

(###### /Outputs/Artifacts/Name)

(Artifacts - outputが含まれている場合は必須)

アクションによって生成されたアーティファクトの名前を指定します。アーティファクト名はワークフロー内で一意である必要があり、英数字 (a～z、A～Z、0～9) とアンダースコア (_) に制限され

ます。スペース、ハイフン (-)、その他の特殊文字は使用できません。引用符を使用して、出力アーティファクト名でスペース、ハイフン、その他の特殊文字を有効にすることはできません。

アーティファクトの詳細については、「」を参照してくださいアーティファクトを使用したワークフロー内のアクション間のデータの共有。

対応する UI: 出力タブ/アーティファクト/アーティファクト名の追加/ビルド

Files

(###### /Outputs/Artifacts/Files)

(Artifacts - outputが含まれている場合は必須)

アクションによって出力されるアーティファクトに CodeCatalyst を含むファイルを指定します。これらのファイルは、実行時にワークフローアクションによって生成され、ソースリポジトリでも使用できます。ファイルパスは、ソースリポジトリまたは前のアクションのアーティファクトに存在し、ソースリポジトリまたはアーティファクトルートを基準にしています。glob パターンを使用してパスを指定できます。例:

- ビルドまたはソースリポジトリの場所のルートにある 1 つのファイルを指定するには、`my-file.jar` を使用します。
- サブディレクトリ内の 1 つのファイルを指定するには、`directory/my-file.jar` または `directory/subdirectory/my-file.jar` を使用します。
- すべてのファイルを指定するには、`"**/*"` を使用します。glob パターン `**` は、任意の数のサブディレクトリにマッチすることを示します。
- `directory` という名前のディレクトリ内のすべてのファイルとディレクトリを指定するには、`"directory/**/*"` を使用します。glob パターン `**` は、任意の数のサブディレクトリにマッチすることを示します。
- `directory` という名前のディレクトリ内のすべてのファイルを指定するが、そのサブディレクトリを含めないようにするには、`"directory/*"` を使用します。

> **Note**
>
> ファイルパスに 1 つ以上のアスタリスク (*) またはその他の特殊文字が含まれている場合は、パスを二重引用符 () で囲みます""。特殊文字の詳細については、「」を参照してください構文のガイドラインと規則。

アーティファクトの詳細については、「」を参照してくださいアーティファクトを使用したワークフロー内のアクション間のデータの共有。

> **Note**
>
> ファイルパスにプレフィックスを追加して、検索するアーティファクトまたはソースを示す必要がある場合があります。詳細については、「ソースリポジトリ内のファイルを参照する」および「アーティファクト内のファイルを参照する」を参照してください。

対応する UI: タブ/アーティファクトを出力/ビルドによって生成されたアーティファクト/ファイルを追加

Variables - output

(###### /Outputs/Variables)

(オプション)

後続のアクションで使用できるように、アクションでエクスポートする変数を指定します。

例を含む変数の詳細については、「」を参照してくださいワークフローでの変数の設定と使用。

対応する UI: 出力タブ/変数/変数の追加

変数名 1

(###### variable-/Outputs/Variablesname-1)

(オプション)

アクションでエクスポートする変数の名前を指定します。この変数は、同じアクションの `Inputs`または `Steps`セクションで既に定義されている必要があります。

例を含む変数の詳細については、「」を参照してくださいワークフローでの変数の設定と使用。

対応する UI: Outputs tab/Variables/Add variable/Name

AutoDiscoverReports

(###### /Outputs/AutoDiscoverReports)

(オプション)

自動検出機能の設定を定義します。

自動検出を有効にすると、 は、アクションに渡Inputsされたすべてのファイルと、アクション自体によって生成されたすべてのファイル CodeCatalyst を検索し、テスト、コードカバレッジ、ソフトウェアコンポジション分析 (SCA) レポートを探します。見つかったレポートごとに、 はレポートに変換 CodeCatalyst します CodeCatalyst 。CodeCatalyst レポートは、 CodeCatalyst サービスに完全に統合され、 CodeCatalyst コンソールから表示および操作できるレポートです。

> **Note**
>
> デフォルトでは、自動検出機能はすべてのファイルを検査します。IncludePaths または ExcludePathsプロパティを使用して、検査するファイルを制限できます。

対応する UI: なし

Enabled

(###### /Outputs/AutoDiscoverReports/Enabled)

(オプション)

自動検出機能を有効または無効にします。

有効な値は `true` または `false` です。

`Enabled` を省略した場合、デフォルトは ですtrue。

対応する UI: 出力タブ/レポート/レポートの自動検出

ReportNamePrefix

(###### /Outputs/AutoDiscoverReports/ReportNamePrefix)

(AutoDiscoverReportsが含まれ、有効になっている場合は必須)

関連するレポートに名前を付けるために、見つかったすべての CodeCatalyst レポートの CodeCatalyst 先頭に プレフィックスを指定します。例えば、プレフィックスとして を指定しAutoDiscovered CodeCatalyst、2 つのテストレポート `TestSuiteOne.xml`と を自動検出すると`TestSuiteTwo.xml`、関連する CodeCatalyst レポートの名前は `AutoDiscoveredTestSuiteOne`と になります`AutoDiscoveredTestSuiteTwo`。

対応する UI: 出力タブ/レポート/レポートの自動検出/レポートプレフィックス

IncludePaths

(###### /Outputs/AutoDiscoverReports/IncludePaths)

または

(###### report-/Outputs/Reports/name-1/IncludePaths)

(AutoDiscoverReportsが含まれ、有効になっている場合、または Reports が含まれている場合は必須)

raw レポートを検索するときに CodeCatalyst に含まれるファイルとファイルパスを指定します。例えば、 を指定すると"/test/report/*"、 は /test/report/* ディレクトリを検索するアクションで使用されるビルドイメージ全体 CodeCatalyst を検索します。そのディレクトリが見つか CodeCatalyst ると、 はそのディレクトリ内のレポートを検索します。

> **Note**
>
> ファイルパスに 1 つ以上のアスタリスク (*) またはその他の特殊文字が含まれている場合は、パスを二重引用符 () で囲みます""。特殊文字の詳細については、「」を参照してください構文のガイドラインと規則。

このプロパティを省略した場合、デフォルトは です。つまり"**/*"、検索にはすべてのパスのすべてのファイルが含まれます。

> **Note**
>
> 手動で設定するレポートの場合、 は単一のファイルに一致する glob パターンIncludePathsである必要があります。

対応する UI:

- 出力タブ/レポート/レポートの自動検出/'Include/exclude paths'/Include paths
- 出力タブ/レポート/レポートを手動で設定/report-name-1/'Include/exclude paths'/Include paths

ExcludePaths

(###### /Outputs/AutoDiscoverReports/ExcludePaths)

または

(###### *report-*/Outputs/Reports/*name-1*/ExcludePaths)

(オプション)

raw レポートを検索するときに が CodeCatalyst 除外するファイルとファイルパスを指定します。例えば、 を指定した場合"/test/my-reports/**/*"、 CodeCatalyst は /test/my-reports/ ディレクトリ内のファイルを検索しません。ディレクトリ内のすべてのファイルを無視するには、glob **/* パターンを使用します。

> **Note**
>
> ファイルパスに 1 つ以上のアスタリスク (*) またはその他の特殊文字が含まれている場合は、パスを二重引用符 () で囲みます""。特殊文字の詳細については、「」を参照してください構文のガイドラインと規則。

対応する UI:

- 出力タブ/レポート/レポートの自動検出/'Include/exclude paths/Exclude paths
- 出力タブ/レポート/レポート/レポート/*report-name-1*/'Include/exclude paths/Exclude paths

SuccessCriteria

(###### /Outputs/AutoDiscoverReports/SuccessCriteria)

または

(###### *report-*/Outputs/Reports/*name-1*/SuccessCriteria)

(オプション)

テスト、コードカバレッジ、ソフトウェア構成分析 (SCA)、静的分析 (SA) レポートの成功基準を指定します。

詳細については、「レポートの成功基準の設定」を参照してください。

対応する UI:

- 出力タブ/レポート/レポートの自動検出/成功基準

- 出力タブ/レポート/レポートを手動で設定/*report-name-1*/成功基準

PassRate

(###### /Outputs/AutoDiscoverReports/SuccessCriteria/PassRate)

または

(###### *report-*/Outputs/Reports/*name-1*/SuccessCriteria/PassRate)

(オプション)

関連する CodeCatalystレポートが合格としてマークされるようにするために合格する必要があるテストレポート内のテストの割合を指定します。有効な値には 10 進数が含まれます。例: 50、60.5。合格率基準はテストレポートにのみ適用されます。テストレポートの詳細については、「」を参照してくださいテストレポート。

対応する UI:

- 出力タブ/レポート/レポートの自動検出/成功基準/合格率
- 出力タブ/レポート/レポートを手動で設定/*report-name-1*/成功基準/合格率

LineCoverage

(###### /Outputs/AutoDiscoverReports/SuccessCriteria/LineCoverage)

または

(###### *report-*/Outputs/Reports/*name-1*/SuccessCriteria/LineCoverage)

(オプション)

コードカバレッジレポートで、関連するレポートが合格としてマークされる CodeCatalyst ためにカバーする必要がある行の割合を指定します。有効な値には 10 進数が含まれます。例: 50、60.5。行カバレッジ基準は、コードカバレッジレポートにのみ適用されます。コードカバレッジレポートの詳細については、「」を参照してくださいコードカバレッジレポート。

対応する UI:

- 出力タブ/レポート/レポートの自動検出/成功基準/ラインカバレッジ

- 出力タブ/レポート/レポートを手動で設定/*report-name-1*/成功基準/ラインカバレッジ

BranchCoverage

(****** /Outputs/AutoDiscoverReports/SuccessCriteria/BranchCoverage)

または

(****** *report-*/Outputs/Reports/*name-1*/SuccessCriteria/BranchCoverage)

(オプション)

コードカバレッジレポートで、関連するレポートを合格としてマーク CodeCatalyst するためにカバーする必要があるブランチの割合を指定します。有効な値には 10 進数が含まれます。例: `50`、`60.5`。ブランチカバレッジ基準は、コードカバレッジレポートにのみ適用されます。コードカバレッジレポートの詳細については、「」を参照してください[コードカバレッジレポート]。

対応する UI:

- 出力タブ/レポート/レポートの自動検出/成功基準/ブランチカバレッジ
- 出力タブ/レポート/レポートを手動で設定/*report-name-1*/成功基準/ブランチカバレッジ

Vulnerabilities

(****** /Outputs/AutoDiscoverReports/SuccessCriteria/Vulnerabilities)

または

(****** *report-*/Outputs/Reports/*name-1*/SuccessCriteria/Vulnerabilities)

(オプション)

SCA レポートで許可される脆弱性の最大数と重要度を指定して、関連する CodeCatalyst レポートが合格としてマークされるようにします。脆弱性を指定するには、以下を指定する必要があります。

- カウントに含める脆弱性の最小重要度。最も重要度の高い値から低い値までの有効な値は、`CRITICAL`、`HIGH`、、`MEDIUM`、`LOW` ですINFORMATIONAL。

  例えば、 を選択するとHIGH、 HIGHと のCRITICAL脆弱性が集計されます。
- 許可する指定された重要度の脆弱性の最大数。この数を超えると、 CodeCatalyst レポートは失敗としてマークされます。有効な値は整数です。

脆弱性基準は SCA レポートにのみ適用されます。SCA レポートの詳細については、「」を参照してくださいソフトウェア構成分析レポート。

最小重要度を指定するには、 `Severity`プロパティを使用します。脆弱性の最大数を指定するには、 `Number`プロパティを使用します。

SCA レポートの詳細については、「」を参照してください品質レポートタイプ。

対応する UI:

- 出力タブ/レポート/レポートの自動検出/成功基準/脆弱性
- 出力タブ/レポート/レポートを手動で設定/*report-name-1*/成功基準/脆弱性

Reports

(*######*/Outputs/Reports)

(オプション)

テストレポートの設定を指定するセクション。

対応する UI: 出力タブ/レポート

report-name-1

(*###### report-*/Outputs/Reports/name-1)

(Reportsが含まれている場合は必須)

raw CodeCatalyst レポートから生成されるレポートに付ける名前。

対応する UI: 出力タブ/レポート/レポートを手動で設定/レポート名

Format

(*###### report-*/Outputs/Reports/*name-1*/Format)

(Reportsが含まれている場合は必須)

レポートに使用するファイル形式を指定します。指定できる値は以下のとおりです。

- テストレポートの場合：

  - Cucumber JSON の場合は、Cucumber (ビジュアルエディタ) または CUCUMBERJSON (YAML エディタ) を指定します。
  - JUnit XML の場合は、JUnit (ビジュアルエディタ) または JUNITXML (YAML エディタ) を指定します。
  - NUnit XML の場合は、NUnit (ビジュアルエディタ) または NUNITXML (YAML エディタ) を指定します。
  - NUnit 3 XML の場合は、NUnit3 (ビジュアルエディタ) または NUNIT3XML (YAML エディタ) を指定します。
  - Visual Studio TRX には、Visual Studio TRX (ビジュアルエディタ) または VISUALSTUDIOTRX (YAML エディタ) を指定します。
  - TestNG XML の場合は、TestNG (ビジュアルエディタ) または TESTNGXML (YAML エディタ) を指定します。
- コードカバレッジレポートの場合：
  - クローバー XML には、クローバー (ビジュアルエディタ) または CLOVERXML (YAML エディタ) を指定します。
  - コベルチュラ XML では、コベルチュラ (ビジュアルエディタ) または COBERTURAXML (YAML エディタ) を指定します。
  - JaCoCo XML の場合は、 JaCoCo (ビジュアルエディタ) または JACOCOXML (YAML エディタ) を指定します。
  - simplecov-json ではなく simplecov によって生成された SimpleCov JSON の場合は、Simplecov (ビジュアルエディタ) または SIMPLECOV (YAML エディタ) を指定します。 https://github.com/vicentllongo/simplecov-json
- ソフトウェアコンポジション分析 (SCA) レポートの場合：
  - SARIF には、SARIF (ビジュアルエディタ) または SARIFSCA (YAML エディタ) を指定します。

対応する UI: 出力タブ/レポート/レポートを手動で設定/レポートの追加/レポート/*#####-1*/レポートタイプとレポート形式

Configuration

(*######* /Configuration)

(必須) アクションの設定プロパティを定義できるセクション。

対応する UI: 設定タブ

Steps

(###### /Configuration/Steps)

(必須)

GitHub Marketplace の GitHub アクションの詳細ページに表示されるアクションコードを指定します。次のガイドラインに従ってコードを追加します。

1. GitHub アクションの steps:セクションのコードを CodeCatalyst ワークフローの Steps:セクションに貼り付けます。コードはダッシュ (-) で始まり、次のようになります。

   GitHub 貼り付けるコード：

   ```
   - name: Lint Code Base
     uses: github/super-linter@v4
     env:
       VALIDATE_ALL_CODEBASE: false
       DEFAULT_BRANCH: master
       GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
   ```

2. 貼り付けたコードを確認し、標準に準拠 CodeCatalystするように必要に応じて変更します。例えば、前述のコードブロックでは、#### のコードを削除し、太字 のコードを追加できます。

   CodeCatalyst ワークフロー yaml:

   ```
   Steps:
      - name: Lint Code Base
        uses: github/super-linter@v4
        env:
          VALIDATE_ALL_CODEBASE: false
          DEFAULT_BRANCH: mastermain
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
   ```

3. GitHub アクションに含まれているが、 steps:セクション内に存在しない追加のコードについては、 CodeCatalystと同等のコードを使用して CodeCatalyst ワークフローに追加します。を確認してワークフロー YAML 定義、 GitHub コードを に移植する方法に関するインサイトを得ることができます CodeCatalyst。詳細な移行手順は、このガイドの対象外です。

GitHub アクションアクションでファイルパスを指定する方法の例を次に示します。

```
Steps:
```

```
  - name: Lint Code Base
    uses: github/super-linter@v4
    ...
  - run: cd /sources/WorkflowSource/MyFolder/  && cat file.txt
  - run: cd /artifacts/MyGitHubAction/MyArtifact/MyFolder/  && cat file2.txt
```

ファイルパスの指定の詳細については、ソースリポジトリ内のファイルを参照する「」および「」を参照してくださいアーティファクト内のファイルを参照する。

対応する UI: 設定タブ/GitHub アクション YAML

ワークフローのコンピューティング環境とランタイム環境の Docker イメージの設定

CodeCatalyst ワークフローでは、 がワークフローアクションを実行 CodeCatalyst するために使用するコンピューティング環境イメージとランタイム環境イメージを指定できます。

コンピューティングとは、ワークフローアクションを実行 CodeCatalyst するために が管理および保守するコンピューティングエンジン (CPU、メモリ、オペレーティングシステム) を指します。

> **Note**
>
> コンピューティングがワークフローのプロパティとして定義されている場合、そのワークフロー内のアクションのプロパティとして定義することはできません。同様に、コンピューティングがアクションのプロパティとして定義されている場合、ワークフローで定義することはできません。

ランタイム環境イメージは、 がワークフローアクションを実行する CodeCatalyst Docker コンテナです。Docker コンテナは、選択したコンピューティングプラットフォーム上で実行され、オペレーティングシステムと、、Node.js AWS CLI、.tar などのワークフローアクションに必要な追加ツールが含まれています。

トピック

- コンピューティングタイプ
- コンピューティングフリート
- オンデマンドフリートのプロパティ

- プロビジョニングされたフリートのプロパティ
- プロビジョニングされたフリートの作成
- プロビジョニングされたフリートの編集
- プロビジョニングされたフリートの削除
- プロビジョニングされたフリートまたはオンデマンドコンピューティングをアクションに割り当てる
- アクション間でのコンピューティングの共有
- ランタイム環境の Docker イメージの指定

コンピューティングタイプ

CodeCatalyst には、次のコンピューティングタイプがあります。

- Amazon EC2
- AWS Lambda

Amazon EC2 は、アクション実行中の柔軟性を最適化し、Lambda はアクションの起動速度を最適化します。Lambda は、起動レイテンシーが低いため、ワークフローアクションの実行の高速化をサポートします。Lambda を使用すると、一般的なランタイムでサーバーレスアプリケーションを構築、テスト、デプロイできる基本的なワークフローを実行できます。これらのランタイムには、Node.js、Python、Java、.NET、Go が含まれます。ただし、Lambda がサポートしていないユースケースがいくつかあり、それらがユーザーに影響を与える場合は、Amazon EC2 コンピューティングタイプを使用します。

- Lambda は、指定されたレジストリからのランタイム環境イメージをサポートしていません。
- Lambda は、ルートアクセス許可を必要とするツールをサポートしていません。yum や などのツールの場合は`rpm`、Amazon EC2 コンピューティングタイプ、またはルートアクセス許可を必要としないその他のツールを使用します。
- Lambda は Docker のビルドまたは実行をサポートしていません。Docker イメージを使用する以下のアクションはサポートされていません: AWS CloudFormation スタックのデプロイ、Amazon ECS へのデプロイ、Amazon S3 公開、 AWS CDK ブートストラップ、 AWS CDK デプロイ、 AWS Lambda 呼び出し、および GitHub アクション。Actions GitHub アクション内で CodeCatalyst実行されている Docker ベースの GitHub アクションは、Lambda コンピューティングでもサポートされていません。Podman など、root 権限を必要としない代替手段を使用できます。

- Lambda は、 外のファイルへの書き込みをサポートしていません/tmp。ワークフローアクションを設定するときに、ツールを再設定して をインストールまたは書き込みできます/tmp。をインストールするビルドアクションがある場合はnpm、必ず にインストールするように設定してください/tmp。
- Lambda は 15 分を超えるランタイムをサポートしていません。

コンピューティングフリート

CodeCatalyst は、次のコンピューティングフリートを提供します。

- オンデマンドフリート
- プロビジョニングされたフリート

オンデマンドフリートでは、ワークフローアクションが開始されると、ワークフローは必要なリソースをプロビジョニングします。アクションが終了すると、マシンは破棄されます。料金は、アクションを実行している分数に対してのみ発生します。オンデマンドフリートはフルマネージド型で、需要の急増にも対応できる自動スケーリング機能を備えています。

CodeCatalyst は、 によって管理される Amazon EC2 を搭載したマシンを含むプロビジョニングされたフリートも提供します CodeCatalyst。プロビジョニングされたフリートでは、ワークフローアクションを実行するように一連の専用マシンを設定します。これらのマシンはアイドル状態のままで、すぐにアクションを処理できます。プロビジョニングされたフリートでは、マシンは常に実行され、プロビジョニングされている限りコストが発生します。

フリートを作成、更新、または削除するには、スペース管理者ロールまたはプロジェクト管理者ロールが必要です。

オンデマンドフリートのプロパティ

CodeCatalyst は、次のオンデマンドフリートを提供します。

名前	オペレーティングシステム	アーキテクチャ	vCPUs	メモリ (GiB)	ディスク容量	サポートされているコンピューティングタイプ
`Linux.Arm64.Large`	Amazon Linux 2	Arm64	2	4	64 GB	Amazon EC2
					10 GB	Lambda
`Linux.Arm64.XLarge`	Amazon Linux 2	Arm64	4	8	128 GB	Amazon EC2
					10 GB	Lambda
`Linux.Arm64.2XLarge`	Amazon Linux 2	Arm64	8	16	128 GB	Amazon EC2
`Linux.x86-64.Large`	Amazon Linux 2	x86-64	2	4	64 GB	Amazon EC2
					10 GB	Lambda
`Linux.x86-64.XLarge`	Amazon Linux 2	x86-64	4	8	128 GB	Amazon EC2
					10 GB	Lambda
`Linux.x86-64.2XLarge`	Amazon Linux 2	x86-64	8	16	128 GB	Amazon EC2

> **Note**
>
> オンデマンドフリートの仕様は、請求階層によって異なります。詳細については、「 の料金」を参照してください。

フリートが選択されていない場合は、 CodeCatalyst を使用しますLinux.x86-64.Large。

プロビジョニングされたフリートのプロパティ

プロビジョニングされたフリートには、次のプロパティが含まれます。

オペレーティングシステム

オペレーティングシステム。使用できるオペレーションシステムは次のとおりです。

- Amazon Linux 2
- Windows Server 2022

> **Note**
>
> Windows フリートはビルドアクションでのみサポートされます。その他のアクションは現在 Windows をサポートしていません。

アーキテクチャ

プロセッサアーキテクチャ。以下のアーキテクチャが利用可能です。

- x86_64
- Arm64

マシンタイプ

各インスタンスのマシンタイプ。次のマシンタイプを使用できます。

vCPUs	メモリ (GiB)	ディスク容量	オペレーティングシステム
2	4	64 GB	Amazon Linux 2
4	8	128 GB	Amazon Linux 2
			Windows Server 2022
8	16	128 GB	Amazon Linux 2

vCPUs	メモリ (GiB)	ディスク容量	オペレーティングシステム
			Windows Server 2022

容量

フリートに割り当てられるマシンの初期数。並列で実行できるアクションの数を定義します。

スケーリングモード

アクションの数がフリート容量を超えた場合の動作を定義します。

オンデマンドで追加の容量をプロビジョニングする

追加のマシンはオンデマンドでセットアップされ、実行中の新しいアクションに応じて自動的にスケールアップされ、アクションが終了するとベース容量にスケールダウンされます。実行中のマシンごとに分単位でのお支払いとなるため、追加のコストが発生する可能性があります。

追加のフリート容量が利用可能になるまで待機する

アクションの実行は、マシンが使用可能になるまでキューに入れられます。これにより、さらにマシンが割り当てられないため、追加のコストが抑えられます。

プロビジョニングされたフリートの作成

プロビジョニングされたフリートを作成するには、以下の手順に従います。

> **Note**
>
> プロビジョニングされたフリートは、2 週間の非アクティブ状態が続くと非アクティブ化されます。再度使用すると自動的に再アクティブ化されますが、この再アクティブ化によりレイテンシーが発生する可能性があります。

プロビジョニングされたフリートを作成するには

1. ナビゲーションペインで CI/CD を選択し、コンピューティング を選択します。
2. プロビジョニングされたフリートの作成 を選択します。

3. プロビジョニングされたフリート名のテキストフィールドに、フリートの名前を入力します。
4. [オペレーティングシステム] ドロップダウンメニューから、オペレーティングシステムを選択します。
5. マシンタイプのドロップダウンメニューから、マシンのマシンタイプを選択します。
6. 容量テキストフィールドに、フリート内のマシンの最大数を入力します。
7. [スケーリングモード] ドロップダウンメニューから、目的のオーバーフロー動作を選択します。フィールドの詳細については、「プロビジョニングされたフリートのプロパティ」を参照してください。
8. [作成] を選択します。

プロビジョニングされたフリートを作成したら、アクションに割り当てる準備が整います。詳細については、「プロビジョニングされたフリートまたはオンデマンドコンピューティングをアクションに割り当てる」を参照してください。

プロビジョニングされたフリートの編集

プロビジョニングされたフリートを編集するには、以下の手順に従います。

> **Note**
>
> プロビジョニングされたフリートは、2 週間の非アクティブ状態が続くと非アクティブ化されます。再度使用すると自動的に再アクティブ化されますが、この再アクティブ化によりレイテンシーが発生する可能性があります。

プロビジョニングされたフリートを編集するには

1. ナビゲーションペインで CI/CD を選択し、コンピューティング を選択します。
2. プロビジョンドフリートリストで、編集するフリートを選択します。
3. [編集] を選択します。
4. 容量テキストフィールドに、フリート内のマシンの最大数を入力します。
5. [スケーリングモード] ドロップダウンメニューから、目的のオーバーフロー動作を選択します。フィールドの詳細については、「プロビジョニングされたフリートのプロパティ」を参照してください。
6. [保存] を選択します。

プロビジョニングされたフリートの削除

プロビジョニングされたフリートを削除するには、以下の手順に従います。

プロビジョニングされたフリートを削除するには

> **Warning**
> プロビジョニングされたフリートを削除する前に、アクションの YAML コードから `Fleet`プロパティを削除して、すべてのアクションから削除します。削除された後もプロビジョニングされたフリートを引き続き参照するアクションは、次回アクションが実行されるときに失敗します。

1. ナビゲーションペインで CI/CD を選択し、コンピューティング を選択します。
2. プロビジョニングされたフリートリストで、削除するフリートを選択します。
3. [削除] を選択します。
4. **`delete`** を入力して削除を確定します。
5. [削除] をクリックします。

プロビジョニングされたフリートまたはオンデマンドコンピューティングをアクションに割り当てる

デフォルトでは、ワークフローアクションは Amazon EC2 `Linux.x86-64.Large` コンピューティングタイプのオンデマンドフリートを使用します。代わりにプロビジョニングされたフリートを使用するか、 などの別のオンデマンドフリートを使用するには`Linux.x86-64.2XLarge`、以下の手順に従います。

Visual

開始する前に

- プロビジョニングされたフリートを割り当てる場合は、まずプロビジョニングされたフリートを作成する必要があります。詳細については、「プロビジョニングされたフリートの作成」を参照してください。

プロビジョニングされたフリートまたは異なるフリートタイプをアクションに割り当てるには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. ビジュアル を選択します。
7. ワークフロー図で、プロビジョニングされたフリートまたは新しいフリートタイプを割り当てるアクションを選択します。
8. [設定] タブを選択します。
9. コンピューティングフリート で、次の操作を行います。

   ワークフローまたはワークフローアクションを実行するマシンまたはフリートを指定します。オンデマンドフリートでは、アクションが開始されると、ワークフローは必要なリソースをプロビジョニングし、アクションが終了するとマシンは破棄されます。オンデマンドフリートの例: `Linux.x86-64.Large`、`Linux.x86-64.XLarge`。オンデマンドフリートの詳細については、「」を参照してくださいオンデマンドフリートのプロパティ。

   プロビジョニングされたフリートでは、ワークフローアクションを実行するように一連の専用マシンを設定します。これらのマシンはアイドル状態のままで、すぐにアクションを処理できます。プロビジョニングされたフリートの詳細については、「」を参照してくださいプロビジョニングされたフリートのプロパティ。

   `Fleet` を省略した場合、デフォルトは ですLinux.x86-64.Large。
10. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
11. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

YAML

開始する前に

- プロビジョニングされたフリートを割り当てる場合は、まずプロビジョニングされたフリートを作成する必要があります。詳細については、「プロビジョニングされたフリートの作成」を参照してください。

プロビジョニングされたフリートまたは異なるフリートタイプをアクションに割り当てるには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。/
5. [編集] を選択します。
6. YAML を選択します。
7. プロビジョニングされたフリートまたは新しいフリートタイプを割り当てるアクションを見つけます。
8. アクションで、 `Compute`プロパティを追加し、 `Fleet`をフリートの名前またはオンデマンドフリートタイプに設定します。詳細については、 ビルドおよびテストアクションの YAML 定義アクションの の `Fleet` プロパティの説明を参照してください。
9. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
10. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

アクション間でのコンピューティングの共有

デフォルトでは、ワークフロー内のアクションはフリート 内の個別のインスタンスで実行されます。この動作は、入力の状態を分離し、予測可能なアクションを提供します。デフォルトの動作では、ファイルや変数などのコンテキストをアクション間で共有するための明示的な設定が必要です。

コンピューティング共有は、ワークフロー内のすべてのアクションを同じインスタンスで実行できる機能です。コンピューティング共有を使用すると、インスタンスのプロビジョニングに費やす時間が

短縮されるため、ワークフローのランタイムが短縮されます。ワークフロー設定を追加しなくても、アクション間でファイル (アーティファクト) を共有することもできます。

コンピューティング共有を使用してワークフローを実行すると、デフォルトまたは指定されたフリートのインスタンスは、そのワークフロー内のすべてのアクションの期間中予約されます。ワークフローの実行が完了すると、インスタンスの予約が解放されます。

トピック

共有コンピューティングで複数のアクションを実行する

ワークフローレベルで定義 YAML の `Compute` 属性を使用して、アクションのフリート共有プロパティとコンピューティング共有プロパティの両方を指定できます。のビジュアルエディタを使用してコンピューティングプロパティを設定することもできます CodeCatalyst。フリートを指定するには、既存のフリートの名前を設定し、コンピューティングタイプを EC2 に設定して、コンピューティング共有を有効にします。

> **Note**
>
> コンピューティング共有は、コンピューティングタイプが EC2 に設定されている場合にのみサポートされ、Windows Server 2022 オペレーティングシステムではサポートされていません。コンピューティングフリート、コンピューティングタイプ、プロパティの詳細については、「」を参照してくださいワークフローのコンピューティング環境とランタイム環境の Docker イメージの設定。

> **Note**
>
> 無料利用枠で、ワークフロー定義 YAML で `Linux.x86-64.XLarge`または `Linux.x86-64.2XLarge`フリートを手動で指定すると、アクションは引き続きデフォルトのフリート () で実行されます`Linux.x86-64.Large`。コンピューティングの可用性と料金の詳細については、階層オプション の表を参照してください。

コンピューティング共有がオンになっている場合、ワークフローソースを含むフォルダはアクション間で自動的にコピーされます。出力アーティファクトを設定し、ワークフロー定義 (YAML ファイル) 全体で入力アーティファクトとして参照する必要はありません。ワークフロー作成者は、コンピューティング共有を使用しない場合と同様に、入力と出力を使用して環境変数をワイヤアップする必要があります。ワークフローソース外のアクション間でフォルダを共有する場合は、ファイルのキャッシュを検討してください。詳細については、「アーティファクトを使用したワークフロー内のアクション間のデータの共有」および「ワークフロー実行間のファイルのキャッシュ」を参照してください。

ワークフロー定義ファイルが存在するソースリポジトリは、ラベル によって識別されますWorkflowSource。コンピューティング共有を使用している間、ワークフローソースは、それを参照する最初のアクションでダウンロードされ、ワークフロー実行の後続のアクションで使用できるようになります。ファイルの追加、変更、削除などのアクションによってワークフローソースを含むフォルダに加えられた変更は、ワークフローの後続のアクションにも表示されます。コンピューティング共有を使用せずに、ワークフローの任意のアクションのワークフローソースフォルダにあるファイルを参照できます。詳細については、「ソースリポジトリ内のファイルを参照する」を参照してください。

> **Note**
>
> コンピューティング共有ワークフローでは、並列アクションを設定できないように、アクションの厳密なシーケンスを指定する必要があります。出力アーティファクトはシーケンス内の任意のアクションで設定できますが、入力アーティファクトはサポートされていません。

コンピューティング共有に関する考慮事項

コンピューティング共有を使用してワークフローを実行して、ワークフローの実行を高速化し、同じインスタンスを使用するワークフロー内のアクション間でコンテキストを共有できます。コンピューティング共有の使用がシナリオに適しているかどうかを判断するには、次の点を考慮してください。

	コンピューティング共有	コンピューティング共有なし
コンピューティングタイプ	Amazon EC2	Amazon EC2、AWS Lambda
インスタンスのプロビジョニング	同じインスタンスで実行されるアクション	アクションは個別のインスタンスで実行されます

	コンピューティング共有	コンピューティング共有なし
オペレーティングシステム	Amazon Linux 2	Amazon Linux 2、Windows Server 2022 (ビルドアクションのみ)
ファイルを参照する	`$CATALYST_SOURCE_DIR_WorkflowSource` , `/sources/WorkflowSource/`	`$CATALYST_SOURCE_DIR_WorkflowSource` , `/sources/WorkflowSource/`
Workflow 構造	アクションは順番にしか実行できません	アクションは並列で実行できます
ワークフローアクション間のデータへのアクセス	キャッシュされたワークフローソースへのアクセス (`WorkflowSource`)	共有アーティファクトのアクセス出力 (追加の設定が必要)

コンピューティング共有を有効にする

次の手順に従って、ワークフローのコンピューティング共有を有効にします。

Visual

ビジュアルエディタを使用してコンピューティング共有を有効にするには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。
5. [編集] を選択します。
6. ビジュアル を選択します。
7. ワークフロープロパティ を選択します。
8. コンピューティングタイプのドロップダウンメニューから、EC2 を選択します。
9. (オプション) コンピューティングフリート - オプションのドロップダウンメニューから、ワークフローアクションの実行に使用するフリートを選択します。オンデマンドフリートを

選択するか、プロビジョニングされたフリートを作成して選択できます。詳細については、「プロビジョニングされたフリートの作成」および「プロビジョニングされたフリートまたはオンデマンドコンピューティングをアクションに割り当てる」を参照してください。

10. トグルを切り替えてコンピューティング共有をオンにし、ワークフロー内のアクションを同じフリートで実行します。
11. (オプション) ワークフローの実行モードを選択します。詳細については、「実行のキューイング動作の設定」を参照してください。
12. コミット を選択し、コミットメッセージを入力し、再度コミット を選択します。

YAML

YAML エディタを使用してコンピューティング共有を有効にするには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。
5. [編集] を選択します。
6. YAML を選択します。
7. コンピューティング共有をオンにして、 SharedInstance フィールドを にTRUE、 Typeを に設定しますEC2。ワークフローアクションの実行に使用するコンピューティングフリートFleetに設定します。オンデマンドフリートを選択するか、プロビジョニングされたフリートを作成して選択できます。詳細については、「プロビジョニングされたフリートの作成」および「プロビジョニングされたフリートまたはオンデマンドコンピューティングをアクションに割り当てる」を参照してください。

   ワークフロー YAML で、次のようなコードを追加します。

```
Name: MyWorkflow
SchemaVersion: "1.0"
Compute: # Define compute configuration.
  Type: EC2
  Fleet: MyFleet # Optionally, choose an on-demand or provisioned fleet.
  SharedInstance: true # Turn on compute sharing. Default is False.
Actions:
  BuildFirst:
```

```
      Identifier: aws/build@v1
      Inputs:
        Sources:
          - WorkflowSource
      Configuration:
        Steps:
          - Run: ...
          ...
```

8. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
9. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

例

トピック

- 例: Amazon S3 Publish

例: Amazon S3 Publish

次のワークフロー例は、Amazon S3 Publish アクションを 2 つの方法で実行する方法を示しています。まず入力アーティファクトを使用し、次にコンピューティング共有を使用します。コンピューティング共有では、キャッシュされた にアクセスできるため、入力アーティファクトは必要ありませんWorkflowSource。さらに、ビルドアクションの出力アーティファクトは不要になりました。S3 Publish アクションは、明示的な DependsOnプロパティを使用してシーケンシャルアクションを維持するように設定されています。S3 Publish アクションを実行するには、ビルドアクションが正常に実行されている必要があります。

- コンピューティング共有を使用しない場合は、入力アーティファクトを使用し、出力を後続のアクションと共有する必要があります。

```
Name: S3PublishUsingInputArtifact
SchemaVersion: "1.0"
Actions:
  Build:
    Identifier: aws/build@v1
    Outputs:
      Artifacts:
        - Name: ArtifactToPublish
```

```
          Files: [output.zip]
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      Steps:
        - Run: ./build.sh # Build script that generates output.zip
  PublishToS3:
    Identifier: aws/s3-publish@v1
    Inputs:
      Artifacts:
      - ArtifactToPublish
    Environment:
      Connections:
        - Role: codecatalyst-deployment-role
          Name: dev-deployment-role
      Name: dev-connection
    Configuration:
      SourcePath: output.zip
      DestinationBucketName: dev-bucket
```

- を SharedInstanceに設定してコンピューティング共有を使用する場合TRUE、同じインスタンスで複数のアクションを実行し、単一のワークフローソースを指定してアーティファクトを共有できます。入力アーティファクトは必須ではなく、指定できません。

```
Name: S3PublishUsingComputeSharing
SchemaVersion: "1.0"
Compute:
  Type: EC2
  Fleet: dev-fleet
  SharedInstance: TRUE
Actions:
  Build:
    Identifier: aws/build@v1
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      Steps:
        - Run: ./build.sh # Build script that generates output.zip
  PublishToS3:
```

```
    Identifier: aws/s3-publish@v1
    DependsOn:
      - Build
    Environment:
      Connections:
        - Role: codecatalyst-deployment-role
          Name: dev-deployment-role
      Name: dev-connection
    Configuration:
      SourcePath: output.zip
      DestinationBucketName: dev-bucket
```

ランタイム環境の Docker イメージの指定

ランタイム環境イメージは、 がワークフローアクションを実行する CodeCatalyst Docker コンテナです。Docker コンテナは、選択したコンピューティングプラットフォーム上で実行され、オペレーティングシステムと、、Node.js AWS CLI、.tar などのワークフローアクションに必要な追加ツールが含まれています。

デフォルトでは、ワークフローアクションは、 によって提供および保守されているアクティブなイメージの 1 つで実行されます CodeCatalyst。ビルドアクションとテストアクションのみがカスタムイメージをサポートします。詳細については、「カスタムランタイム環境の Docker イメージを アクションに割り当てる」を参照してください。

トピック

- アクティブなイメージ
- アクティブなイメージに必要なツールが含まれていない場合はどうなりますか？
- カスタムランタイム環境の Docker イメージを アクションに割り当てる
- 例

アクティブなイメージ

アクティブイメージは、 によって完全にサポートされ CodeCatalyst 、プリインストールされたツールを含むランタイム環境イメージです。現在、アクティブなイメージには 2 つのセットがあります。1 つは 2024 年 3 月にリリースされ、もう 1 つは 2022 年 11 月にリリースされています。

アクションが 2024 年 3 月イメージを使用するか 2022 年 11 月イメージを使用するかは、アクションによって異なります。

- 2024 年 3 月 26 日以降にワークフローに追加されるビルドおよびテストアクションには、2024 年 3 月イメージ を明示的に指定する Containerセクションが YAML 定義に含まれます。オプションで Containerセクションを削除して、2022 年 11 月のイメージ に戻すことができます。
- 2024 年 3 月 26 日より前にワークフローに追加されたビルドおよびテストアクションには、YAML 定義に Containerセクションが含まれず、その結果、2022 年 11 月のイメージ が使用されます。2022 年 11 月のイメージを保持することも、アップグレードすることもできます。イメージをアップグレードするには、ビジュアルエディタで アクションを開き、設定タブを選択し、ランタイム環境の docker イメージドロップダウンリストから 2024 年 3 月イメージを選択します。この選択により、適切な 2024 年 3 月の画像が入力されているアクションの YAML 定義にContainerセクションが追加されます。
- 他のすべてのアクションでは、ワークフローに追加された日時に関係なく、2022 年 11 月のイメージが使用されます。これらのアクションを 2024 年 3 月イメージを使用するようにアップグレードすることはできません。

トピック

- 2024 年 3 月の画像
- 2022 年 11 月の画像

2024 年 3 月の画像

2024 年 3 月のイメージは、 が提供する最新のイメージです CodeCatalyst。2024 年 3 月のイメージは、コンピューティングタイプとフリートの組み合わせごとに 1 つあります。

次の表は、2024 年 3 月の各イメージにインストールされたツールを示しています。

2024 年 3 月のイメージツール

ツール	Linux x86_64 用 CodeCatalyst Amazon EC2 - **CodeCatalystLinux_x86_64:2024_03**	CodeCatalyst Linux x86_64 用 Lambda - **CodeCatalystLinuxLambda_x86_64:2024_03**	Linux Arm64 用 CodeCatalyst Amazon EC2 - **CodeCatalystLinux_Arm64:2024_03**	CodeCatalyst Arm64 用 L - **CodeCatystLinuxambda_Ar64:2024_**
AWS CLI	2.15.17	2.15.17	2.15.17	2.15.17
AWS コパイロット CLI	1.32.1	1.32.1	1.32.1	1.32.1
Docker	24.0.9	該当なし	24.0.9	該当なし
Docker Compose	2.23.3	該当なし	2.23.3	該当なし
Git	2.43.0	2.43.0	2.43.0	2.43.0
Go	1.21.5	1.21.5	1.21.5	1.21.5
Gradle	8.5	8.5	8.5	8.5
Java	Corretto17	Corretto17	Corretto17	Corretto17
Maven	3.9.6	3.9.6	3.9.6	3.9.6
Node.js	18.19.0	18.19.0	18.19.0	18.19.0
npm	10.2.3	10.2.3	10.2.3	10.2.3
Python	3.9.18	3.9.18	3.9.18	3.9.18
Python3	3.11.6	3.11.6	3.11.6	3.11.6
pip	22.3.1	22.3.1	22.3.1	22.3.1
.NET	8.0.100	8.0.100	8.0.100	8.0.100

2022 年 11 月の画像

2022 年 11 月のイメージは、コンピューティングタイプとフリートの組み合わせごとに 1 つあります。プロビジョニングされたフリート を設定している場合、ビルドアクションで 2022 年 11 月の Windows イメージも使用できます。

次の表は、2022 年 11 月の各イメージにインストールされているツールを示しています。

2022 年 11 月のイメージツール

ツール	Linux x86_64 用 CodeCatalyst Amazon EC2 - **CodeCatalystLinux_x86_64:2022_11**	CodeCatalyst Linux x86_64 用 Lambda - **CodeCatalystLinuxLambda_x86_64:2022_11**	Linux Arm64 用 CodeCatalyst Amazon EC2 - **CodeCatalystLinux_Arm64:2022_11**	CodeCatalyst Arm64 用 L - **CodeCatalystLinuxLambda_Arm64:2022_**
AWS CLI	2.15.17	2.15.17	2.15.17	2.15.17
AWS コパイロット CLI	0.6.0	0.6.0	該当なし	該当なし
Docker	23.01	該当なし	23.0.1	該当なし
Docker Compose	2.16.0	該当なし	2.16.0	該当なし
Git	2.40.0	2.40.0	2.39.2	2.39.2
Go	1.20.2	1.20.2	1.20.1	1.20.1
Gradle	8.0.2	8.0.2	8.0.1	8.0.1
Java	Corretto17	Corretto17	Corretto17	Corretto17
Maven	3.9.4	3.9.4	3.9.0	3.9.0
Node.js	16.20.2	16.20.2	16.19.1	16.14.2
npm	8.19.4	8.19.4	8.19.3	8.5.0
Python	3.9.15	2.7.18	3.11.2	2.7.18

ツール	Linux x86_64 用 CodeCatalyst Amazon EC2 - **CodeCatalystLinux_x86_64:2022_11**	CodeCatalyst Linux x86_64 用 Lambda - **CodeCatalystLinuxLambda_x86_64:2022_11**	Linux Arm64 用 CodeCatalyst Amazon EC2 - **CodeCatalystLinux_Arm64:2022_11**	CodeCatalyst Arm64 用 L - **CodeCatystLinuxambda_Ar64:2022_**
Python3	該当なし	3.9.15	該当なし	3.11.2
pip	22.2.2	22.2.2	23.0.1	23.0.1
.NET	6.0.407	6.0.407	6.0.406	6.0.406

アクティブなイメージに必要なツールが含まれていない場合はどうなりますか？

が提供するアクティブなイメージに、必要なツール CodeCatalyst が含まれていない場合は、いくつかのオプションがあります。

- 必要なツールを含むカスタムランタイム環境の Docker イメージを提供できます。詳細については、「カスタムランタイム環境の Docker イメージを アクションに割り当てる」を参照してください。

  > **Note**
  >
  > カスタムランタイム環境の Docker イメージを提供する場合は、カスタムイメージに Git がインストールされていることを確認してください。

- ワークフローのビルドまたはテストアクションに必要なツールをインストールできます。

  例えば、ビルドまたはテストアクションの YAML コードの `Steps`セクションに次の手順を含めることができます。

```
Configuration:
  Steps:
    - Run: ./setup-script
```

次に*#setup-script* 命令は次のスクリプトを実行して Node パッケージマネージャー (npm) をインストールします。

```
#!/usr/bin/env bash
echo "Setting up environment"

touch ~/.bashrc
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.38.0/install.sh | bash
source ~/.bashrc
nvm install v16.1.0
source ~/.bashrc
```

ビルドアクション YAML の詳細については、「」を参照してくださいビルドおよびテストアクションの YAML 定義。

カスタムランタイム環境の Docker イメージを アクションに割り当てる

が提供するアクティブイメージを使用しない場合は CodeCatalyst、カスタムランタイム環境の Docker イメージを指定できます。カスタムイメージを提供する場合は、そのイメージに Git がインストールされていることを確認します。イメージは、Docker Hub、Amazon Elastic Container Registry、または任意のパブリックリポジトリに格納できます。

カスタム Docker イメージを作成する方法については、Docker ドキュメントの「アプリケーションをコンテナ化する」を参照してください。

カスタムランタイム環境の Docker イメージを アクションに割り当てるには、以下の手順に従います。イメージを指定すると、 はアクションの開始時にそのイメージをコンピューティングプラットフォームに CodeCatalyst デプロイします。

> **Note**
>
> 次のアクションは、カスタムランタイム環境 Docker イメージをサポートしていません: AWS CloudFormation スタックのデプロイ 、ECS へのデプロイ 、およびGitHub アクション 。カスタムランタイム環境 Docker イメージは、Lambda コンピューティングタイプもサポートしていません。

Visual

ビジュアルエディタを使用してカスタムランタイム環境の Docker イメージを割り当てるには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
3. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
4. [編集] を選択します。
5. ビジュアル を選択します。
6. ワークフロー図で、カスタムランタイム環境の Docker イメージを使用するアクションを選択します。
7. [設定] タブを選択します。
8. 下部の次のフィールドに入力します。

   ランタイム環境の Docker イメージ - オプション

   イメージが保存されているレジストリを指定します。有効な値を次に示します。

   - `CODECATALYST` (YAML エディタ)

     イメージは CodeCatalyst レジストリに保存されます。
   - Docker Hub (ビジュアルエディタ) または `DockerHub` (YAML エディタ)

     イメージは Docker Hub イメージレジストリに保存されます。
   - その他のレジストリ (ビジュアルエディタ) または `Other` (YAML エディタ)

     イメージはカスタムイメージレジストリに保存されます。公開されているレジストリを使用することができます。
   - Amazon Elastic Container Registry (ビジュアルエディタ) または `ECR` (YAML エディタ)

     イメージは Amazon Elastic Container Registry イメージリポジトリに保存されます。Amazon ECR リポジトリでイメージを使用するには、このアクションで Amazon ECR にアクセスする必要があります。このアクセスを有効にするには、次のアクセス許可とカスタム信頼ポリシーを含む IAM ロールを作成する必要があります。(既存のロールを変更して、必要に応じてアクセス許可とポリシーを含めることができます)。

IAM ロールには、ロールポリシーに次のアクセス許可を含める必要があります。

- ecr:BatchCheckLayerAvailability
- ecr:BatchGetImage
- ecr:GetAuthorizationToken
- ecr:GetDownloadUrlForLayer

IAM ロールには、次のカスタム信頼ポリシーを含める必要があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                 ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

IAM ロールの作成の詳細については、「IAM ユーザーガイド」の「カスタム信頼ポリシーを使用したロールの作成 (コンソール) 」を参照してください。

ロールを作成したら、 環境を介して アクションに割り当てる必要があります。詳細については、「環境、アカウント接続、IAM ロールをワークフローアクションに関連付ける」を参照してください。

ECR イメージ URL、Docker Hub イメージ、またはイメージ URL

次のいずれかを指定します。

- CODECATALYST レジストリを使用している場合は、イメージ を次のアクティブなイメージ のいずれかに設定します。

- `CodeCatalystLinux_x86_64:2024_03`
- `CodeCatalystLinux_x86_64:2022_11`
- `CodeCatalystLinux_Arm64:2024_03`
- `CodeCatalystLinux_Arm64:2022_11`
- `CodeCatalystLinuxLambda_x86_64:2024_03`
- `CodeCatalystLinuxLambda_x86_64:2022_11`
- `CodeCatalystLinuxLambda_Arm64:2024_03`
- `CodeCatalystLinuxLambda_Arm64:2022_11`
- `CodeCatalystWindows_x86_64:2022_11`

- Docker Hub レジストリを使用している場合は、イメージを Docker Hub イメージ名とオプションのタグに設定します。

  例: `postgres:latest`

- Amazon ECR レジストリを使用している場合は、イメージを Amazon ECR レジストリ URI に設定します。

  例: `111122223333.dkr.ecr.us-west-2.amazonaws.com/codecatalyst-ecs-image-repo`

- カスタムレジストリを使用している場合は、イメージをカスタムレジストリで期待される値に設定します。

9. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
10. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

YAML

YAML エディタを使用してカスタムランタイム環境 Docker イメージを割り当てるには

1. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
2. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
3. [編集] を選択します。
4. YAML を選択します。

5. ランタイム環境 Docker イメージを割り当てるアクションを見つけます。
6. アクションで、 Containerセクションと基盤となる Registryプロパティと Imageプロパティを追加します。詳細については、「」のContainerアクション「」、Registryおよびアクションの Imageプロパティの説明を参照してください。
7. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
8. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

例

次の例は、カスタムランタイム環境の Docker イメージをワークフロー定義ファイルの アクションに割り当てる方法を示しています。

トピック

- 例: カスタムランタイム環境の Docker イメージを使用して Amazon ECR で Node.js 18 のサポートを追加する
- 例: カスタムランタイム環境の Docker イメージを使用して、Docker Hub で Node.js 18 のサポートを追加する

例: カスタムランタイム環境の Docker イメージを使用して Amazon ECR で Node.js 18 のサポートを追加する

次の例は、カスタムランタイム環境の Docker イメージを使用して、Amazon ECR で Node.js 18 のサポートを追加する方法を示しています。

```
Configuration:
  Container:
    Registry: ECR
    Image: public.ecr.aws/amazonlinux/amazonlinux:2023
```

例: カスタムランタイム環境の Docker イメージを使用して、Docker Hub で Node.js 18 のサポートを追加する

次の例は、カスタムランタイム環境の Docker イメージを使用して、Docker Hub で Node.js 18 のサポートを追加する方法を示しています。

```
Configuration:
```

```
  Container:
    Registry: DockerHub
    Image: node:18.18.2
```

ワークフローをソースリポジトリに接続する

ソース は、入力ソース とも呼ばれ、オペレーションの実行に必要なファイルを取得するためにワークフローアクションが接続するソースリポジトリです。例えば、ワークフローアクションをソースリポジトリに接続して、アプリケーションを構築するためにアプリケーションソースファイルを取得できます。

CodeCatalyst ワークフローは、次のソースをサポートします。

- CodeCatalyst ソースリポジトリ – 詳細については、「」を参照してくださいでソースリポジトリを使用してコードを保存し、共同作業する CodeCatalyst。
- GitHub および Bitbucket リポジトリ – 詳細については、「」を参照してくださいで拡張機能を使用してプロジェクトに機能を追加する CodeCatalyst。

トピック

- ワークフロー定義ファイルを保存するソースの指定
- ワークフローアクションが使用するソースの指定
- ソースリポジトリ内のファイルを参照する
- ソースによって生成される変数 (BranchName「」およびCommidId「」)

ワークフロー定義ファイルを保存するソースの指定

以下の手順を使用して、ワークフロー定義ファイルを保存する CodeCatalyst ソースリポジトリを指定します。 GitHub または Bitbucket リポジトリを指定する場合は、代わりに「」を参照してくださいで拡張機能を使用してプロジェクトに機能を追加する CodeCatalyst。

ワークフロー定義ファイルが存在するソースリポジトリは、ラベル で識別されますWorkflowSource。

> **Note**
>
> ワークフロー定義ファイルを最初にコミットするときに、ワークフロー定義ファイルが存在するソースリポジトリを指定します。このコミット後、リポジトリとワークフロー定義ファ

イルは永続的にリンクされます。最初のコミット後にリポジトリを変更する唯一の方法は、別のリポジトリでワークフローを再作成することです。

ワークフロー定義ファイルを保存するソースリポジトリを指定するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの作成 を選択し、ワークフローを作成します。詳細については、「ビジュアルエディタを使用してワークフローを作成するには」を参照してください。

   ワークフロー作成プロセス中に、ワークフロー定義ファイルを保存するリポジトリを指定するCodeCatalystように求められます。

ワークフローアクションが使用するソースの指定

以下の手順を使用して、ワークフローアクションで使用するソースリポジトリを指定します。起動時に、 アクションは、設定されたソースリポジトリのファイルをアーティファクトにバンドルし、アクションが実行されているランタイム環境 Docker イメージにアーティファクトをダウンロードし、ダウンロードしたファイルを使用して処理を完了します。

> **Note**
>
> 現在、ワークフローアクション内で指定できるソースリポジトリは 1 つだけです。これは、ワークフロー定義ファイルが存在するソースリポジトリです (`.codecatalyst/workflows/` ディレクトリ内) 。このソースリポジトリはラベル で表されますWorkflowSource。

Visual

アクションが使用するソースリポジトリを指定するには (ビジュアルエディタ)

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。

4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. ビジュアル を選択します。
7. ワークフロー図で、ソースを指定するアクションを選択します。
8. 入力 を選択します。
9. ソース - オプションで以下を実行します。

   アクションに必要なソースリポジトリを表すラベルを指定します。現在、サポートされているラベルは のみです。これは`WorkflowSource`、ワークフロー定義ファイルが保存されているソースリポジトリを表します。

   ソースを省略する場合は、 で少なくとも 1 つの入力アーティファクトを指定する必要があります*`action-name`*`/Inputs/Artifacts`。

   sources の詳細については、「ワークフローをソースリポジトリに接続する」を参照してください。
10. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
11. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

YAML

アクションが使用するソースリポジトリを指定するには (YAML エディタ)

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. YAML を選択します。
7. アクションで、次のようなコードを追加します。

```
action-name:
 Inputs:
   Sources:
     - WorkflowSource
```

詳細については、 ワークフロー YAML 定義アクションの Sourcesプロパティの説明を参照してください。

8. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
9. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

ソースリポジトリ内のファイルを参照する

ソースリポジトリにあるファイルがあり、いずれかのワークフローアクションでこれらのファイルを参照する必要がある場合は、次の手順を実行します。

> **Note**
>
> 「アーティファクト内のファイルを参照する」も参照してください。

ソースリポジトリ内のファイルを参照するには

- ファイルを参照するアクションで、次のようなコードを追加します。

```
Actions:
  My-action:
    Inputs:
      Sources:
        - WorkflowSource
      Configuration:
        Steps:
        - run: cd my-app && cat file1.jar
```

前のコードでは、 アクションはWorkflowSourceソースリポジトリのルートにある my-app ディレクトリを検索して、 file1.jar ファイルを見つけて表示します。

ソースによって生成される変数 (BranchName「」およびCommidId「」)

CodeCatalyst ソースは、ワークフローの実行時にBranchName「」変数とCommitId「」変数を生成して設定します。これらは事前定義された変数 と呼ばれます。これらの変数の詳細については、次の表を参照してください。

ワークフローでこれらの変数を参照する方法については、「」を参照してください事前定義された変数の使用。

キー	値
CommitId	ワークフロー実行の開始時のリポジトリの状態を表すコミット ID。 例: `example3819261db00a3ab59468c8b` 「例：「」のCommitId事前定義された変数を参照する」も参照してください。
BranchName	ワークフローの実行が開始されたブランチの名前。 例: `main`、`feature/branch` 、`test-LiJuan` 「例：「」のBranchName事前定義された変数を参照する」も参照してください。

ワークフローを使用したパッケージの公開とインポート

パッケージは、ソフトウェアのインストールと依存関係の解決に必要なソフトウェアとメタデータの両方を含むバンドルです。 は npm パッケージ形式 CodeCatalyst をサポートします。

パッケージは以下で構成されます。

- 名前 (例えば、 `webpack`は一般的な npm パッケージの名前)
- オプションの名前空間 (など`@types`@types/node)

- 一連のバージョン (、、 1.0.0など1.0.11.0.2)
- パッケージレベルのメタデータ (npm dist タグなど)

では CodeCatalyst、ワークフロー内のパッケージリポジトリにパッケージを公開し、パッケージリポジトリから CodeCatalyst パッケージを使用することができます。パッケージリポジトリを使用して CodeCatalystビルドまたはテストアクションを設定して、指定されたリポジトリからパッケージをプッシュおよびプルするようにアクションの npm クライアントを自動的に設定できます。

パッケージの詳細については、「」を参照してくださいでソフトウェアパッケージを公開および共有する CodeCatalyst。

> **Note**
>
> 現在、ビルドアクションとテストアクションは CodeCatalyst パッケージリポジトリをサポートしています。

トピック

- ワークフローでの CodeCatalyst パッケージリポジトリの指定
- ワークフローでのパッケージリポジトリの指定例

ワークフローでの CodeCatalyst パッケージリポジトリの指定

では CodeCatalyst、ワークフローのビルドアクションとテストアクションに CodeCatalyst パッケージリポジトリを追加できます。パッケージリポジトリは、npm などのパッケージ形式で設定する必要があります。選択したパッケージリポジトリに一連のスコープを含めることもできます。

以下の手順を使用して、ワークフローアクションで使用するパッケージ設定を指定します。

Visual

アクションが使用するパッケージ設定を指定するには (ビジュアルエディタ)

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。

4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. ビジュアル を選択します。
7. ワークフロー図で、パッケージリポジトリで設定するアクションを選択します。
8. パッケージ を選択します。
9. 設定の追加ドロップダウンメニューから、ワークフローアクションで使用するパッケージ設定を選択します。
10. パッケージリポジトリの追加 を選択します。
11. パッケージリポジトリのドロップダウンメニューで、アクションで使用する CodeCatalyst パッケージリポジトリの名前を指定します。

    パッケージリポジトリの詳細については、「」を参照してくださいパッケージリポジトリ。
12. (オプション) スコープ - オプション で、パッケージレジストリで定義するスコープのシーケンスを指定します。

    スコープを定義する場合、指定されたパッケージリポジトリは、リストされているすべてのスコープのレジストリとして設定されます。スコープを持つパッケージが npm クライアントを介してリクエストされた場合、デフォルトではなくそのリポジトリが使用されます。各スコープ名には「@」というプレフィックスを付ける必要があります。

    Scopes を省略すると、指定されたパッケージリポジトリが、 アクションで使用されるすべてのパッケージのデフォルトレジストリとして設定されます。

    スコープの詳細については、パッケージ名前空間「」および「スコープ付きパッケージ」を参照してください。
13. [追加] を選択します。
14. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
15. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

YAML

アクションが使用するパッケージ設定を指定するには (YAML エディタ)

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. YAML を選択します。
7. アクションで、次のようなコードを追加します。

```
action-name:
 Configuration:
    Packages:
        NpmConfiguration:
          PackageRegistries:
            - PackagesRepository: package-repository
              Scopes:
                - "@scope"
```

詳細については、ビルドおよびテストアクションの YAML 定義アクションの Packagesプロパティの説明を参照してください。

8. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
9. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

ワークフローでのパッケージリポジトリの指定例

次の例は、ワークフロー定義ファイル内のパッケージを参照する方法を示しています。

トピック

- 例: を使用したパッケージの定義 NpmConfiguration
- 例: デフォルトレジストリの上書き

- 例: パッケージレジストリ内のスコープの上書き

例: を使用したパッケージの定義 **NpmConfiguration**

次の例は、ワークフロー定義ファイルNpmConfigurationで を使用してパッケージを定義する方法を示しています。

```
Actions:
  Build:
  Identifier: aws/build-beta@v1
  Configuration:
    Packages:
        NpmConfiguration:
          PackageRegistries:
            - PackagesRepository: main-repo
            - PackagesRepository: scoped-repo
              Scopes:
                - "@scope1"
```

この例では、npm クライアントを次のように設定します。

```
default: main-repo
@scope1: scoped-repo
```

この例では、2 つのリポジトリが定義されています。デフォルトのレジストリは、スコープなしで定義されているmain-repoとおりに設定されます。スコープ@scope1は PackageRegistriesの で設定されますscoped-repo。

例: デフォルトレジストリの上書き

次の例は、デフォルトのレジストリを上書きする方法を示しています。

```
NpmConfiguration:
  PackageRegistries:
    - PackagesRepository: my-repo-1
    - PackagesRepository: my-repo-2
    - PackagesRepository: my-repo-3
```

この例では、npm クライアントを次のように設定します。

```
default: my-repo-3
```

複数のデフォルトリポジトリを指定すると、最後のリポジトリが優先されます。この例では、リストされている最後のリポジトリは です。つまりmy-repo-3、npm は に接続しますmy-repo-3。これにより、リポジトリ my-repo-1および が上書きされますmy-repo-2。

例: パッケージレジストリ内のスコープの上書き

次の例は、パッケージレジストリでスコープを上書きする方法を示しています。

```
NpmConfiguration:
  PackageRegistries:
    - PackagesRepository: my-default-repo
    - PackagesRepository: my-repo-1
      Scopes:
        - "@scope1"
        - "@scope2"
    - PackagesRepository: my-repo-2
      Scopes:
        - "@scope2"
```

この例では、npm クライアントを次のように設定します。

```
default: my-default-repo
@scope1: my-repo-1
@scope2: my-repo-2
```

スコープの上書きを含めると、最後のリポジトリが優先されます。この例では、スコープが最後に設定されたの@scope2は の PackageRegistries ですmy-repo-2。これにより、 に@scope2設定されたスコープが上書きされますmy-repo-1。

ワークフローを使用した AWS Lambda 関数の呼び出し

このセクションでは、 CodeCatalyst ワークフローを使用して AWS Lambda 関数を呼び出す方法について説明します。これを実現するには、ワークフローAWS Lambda に呼び出しアクションを追加する必要があります。AWS Lambda 呼び出しアクションは、指定した Lambda 関数を呼び出します。

関数の呼び出しに加えて、AWS Lambda 呼び出しアクションは、Lambda 関数から受信したレスポンスペイロードの各トップレベルキーをワークフロー出力変数 に変換します。これらの変数は、

後続のワークフローアクションで参照できます。すべての最上位キーを変数に変換しない場合は、フィルターを使用して正確なキーを指定できます。詳細については、「」のResponseFilters「プロパティの説明」を参照してくださいAWS Lambda 「呼び出し」アクション YAML 定義。

このアクションを使用するタイミング

Lambda 関数にカプセル化され、Lambda 関数によって実行される機能をワークフローに追加する場合は、このアクションを使用します。

例えば、アプリケーションの構築を開始する前に、ワークフローBuild startedから Slack チャネルに通知を送信したい場合があります。この場合、ワークフローには Lambda をAWS Lambda 呼び出して Slack 通知を送信する呼び出しアクションと、アプリケーションを構築するためのビルドアクションが含まれます。

別の例として、デプロイ前にワークフローでアプリケーションの脆弱性スキャンを実行したい場合があります。この場合、ビルドアクションを使用してアプリケーションを構築し、AWS Lambda 呼び出しアクションを使用して Lambda を呼び出して脆弱性をスキャンし、デプロイアクションを使用してスキャンしたアプリケーションをデプロイします。

トピック

- Lambda 関数を呼び出すワークフローの例
- AWS Lambda 「呼び出し」アクションの追加
- AWS Lambda 「呼び出し」アクションによって生成される変数
- AWS Lambda 「呼び出し」アクション YAML 定義

Lambda 関数を呼び出すワークフローの例

> **Note**
>
> 次のワークフローの例は例示のみを目的としており、正しく機能するためには追加のセットアップ作業が必要です。これは、ワークフローがAWS Lambda 呼び出しアクションで設定されている場合のワークフローの例を示すことを目的としています。

次のワークフローには、 AWS Lambda 呼び出しアクションとデプロイアクションが含まれています。ワークフローは、デプロイが開始されたことを示す Slack 通知を送信し、 AWS CloudFormation

テンプレート AWS を使用して にアプリケーションをデプロイします。ワークフローは、順番に実行される次の構成要素で構成されます。

- トリガー — このトリガーは、変更をソースリポジトリにプッシュすると、ワークフローの実行を自動的に開始します。トリガーについての詳細は、「トリガーを使用したワークフローの自動実行の開始」を参照してください。
- AWS Lambda 呼び出しアクション (LambdaNotify) – トリガー時に、このアクションは指定された AWS アカウントとリージョン (my-aws-account、、) で Notify-Start Lambda 関数を呼び出しますus-west-2。呼び出し時に、Lambda 関数はデプロイが開始されたことを示す Slack 通知を送信します。
- AWS CloudFormation スタックのデプロイアクション (Deploy) — AWS Lambda 呼び出しアクションが完了すると、スタックのデプロイ AWS CloudFormation アクションはテンプレート (cfn-template.yml) を実行してアプリケーションスタックをデプロイします。 AWS CloudFormation スタックのデプロイアクションの詳細については、「」を参照してくださいワークフローを使用した AWS CloudFormation スタックのデプロイ。

```
Name: codecatalyst-lamda-invoke-workflow
SchemaVersion: 1.0

Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  LambdaNotify:
    Identifier: aws/lambda-invoke@v1
    Environment:
      Name: my-production-environment
      Connections:
        - Name: my-aws-account
          Role: codecatalyst-lambda-invoke-role
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      Function: Notify-Start
      AWSRegion: us-west-2

  Deploy:
    Identifier: aws/cfn-deploy@v1
```

```
    Environment:
      Name: my-production-environment
      Connections:
        - Name: my-aws-account
          Role: codecatalyst-deploy-role
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      name: my-application-stack
      region: us-west-2
      role-arn: arn:aws:iam::111122223333:role/StackRole
      template: ./cfn-template.yml
      capabilities: CAPABILITY_IAM,CAPABILITY_AUTO_EXPAND
```

AWS Lambda 「呼び出し」アクションの追加

次の手順を使用して、AWS Lambda 呼び出しアクションをワークフローに追加します。

前提条件

開始する前に、 AWS Lambda 関数と関連する Lambda 実行ロールが で使用可能であることを確認してください AWS。詳細については、「 AWS Lambda デベロッパーガイド」の「Lambda 実行ロール」トピックを参照してください。

Visual

ビジュアルエディタを使用してAWS Lambda 「呼び出し」アクションを追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. ビジュアル を選択します。
7. 左上で + Actions を選択してアクションカタログを開きます。
8. ドロップダウンリストから Amazon CodeCatalystを選択します。

9. AWS Lambda 呼び出しアクションを検索し、次のいずれかを実行します。

   - プラス記号 (+) を選択してワークフロー図にアクションを追加し、設定ペインを開きます。

     または

   - AWS Lambda を呼び出すを選択します。アクションの詳細ダイアログボックスが表示されます。このダイアログボックスで、次の操作を行います。
     - (オプション) ソースを表示 を選択して、アクションのソースコード を表示します。
     - ワークフローに追加 を選択して、ワークフロー図にアクションを追加し、その設定ペインを開きます。

10. Inputs 、Configuration 、および Outputs タブで、必要に応じてフィールドに入力します。各フィールドの説明については、「」を参照してくださいAWS Lambda 「呼び出し」アクション YAML 定義。このリファレンスでは、YAML エディタとビジュアルエディタの両方に表示される各フィールド (および対応する YAML プロパティ値) に関する詳細情報を提供します。
11. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
12. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

YAML

YAML エディタを使用してAWS Lambda 「呼び出し」アクションを追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. YAML を選択します。
7. 左上で + Actions を選択してアクションカタログを開きます。
8. ドロップダウンリストから Amazon CodeCatalystを選択します。
9. AWS Lambda 呼び出しアクションを検索し、次のいずれかを実行します。

- プラス記号 (+) を選択してワークフロー図にアクションを追加し、その設定ペインを開きます。

  または

- AWS Lambda を呼び出すを選択します。アクションの詳細ダイアログボックスが表示されます。このダイアログボックスで、次の操作を行います。
  - (オプション) ソースを表示 を選択して、アクションのソースコード を表示します。
  - ワークフローに追加 を選択して、ワークフロー図にアクションを追加し、その設定ペインを開きます。

10. 必要に応じて YAML コードのプロパティを変更します。使用可能な各プロパティの説明は、「」に記載されていますAWS Lambda 「呼び出し」アクション YAML 定義。
11. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
12. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

AWS Lambda 「呼び出し」アクションによって生成される変数

デフォルトでは、AWS Lambda 呼び出しアクションは Lambda レスポンスペイロードの最上位キーごとに 1 つの変数を生成します。

例えば、レスポンスペイロードは次のようになります。

```
responsePayload = {
  "name": "Saanvi",
  "location": "Seattle",
  "department": {
    "company": "Amazon",
    "team": "AWS"
  }
}
```

アクションは、次の変数を生成します。

キー	値
name	Saanvi

キー	値
ロケーション	Seattle
department	{"company": "Amazon", "team": "AWS"}

> **Note**
>
> `ResponseFilters` YAML プロパティを使用して、生成される変数を変更できます。詳細については、AWS Lambda 「呼び出し」アクション YAML 定義 の「ResponseFilters」を参照してください。

実行時にAWS Lambda 「呼び出し」アクションによって生成および設定された変数は、事前定義された変数 と呼ばれます。

ワークフローでこれらの変数を参照する方法については、「」を参照してください事前定義された変数の使用。

AWS Lambda 「呼び出し」アクション YAML 定義

以下は、AWS Lambda 呼び出しアクションの YAML 定義です。このアクションの使用方法については、「」を参照してくださいワークフローを使用した AWS Lambda 関数の呼び出し。

このアクション定義は、より広範なワークフロー定義ファイル内のセクションとして存在します。ファイルの詳細については、「ワークフロー YAML 定義」を参照してください。

> **Note**
>
> 後続の YAML プロパティのほとんどには、対応する UI 要素がビジュアルエディタにあります。UI 要素を検索するには、Ctrl+F を使用します。 要素は、関連付けられた YAML プロパティとともに一覧表示されます。

```
# The workflow definition starts here.
# See ######### for details.

Name: MyWorkflow
SchemaVersion: 1.0
```

```
Actions:

# The action definition starts here.
  LambdaInvoke_nn:
    Identifier: aws/lambda-invoke@v1
    DependsOn:
      - dependent-action
    Compute:
      Type: EC2 | Lambda
      Fleet: fleet-name
    Timeout: timeout-minutes
    Inputs:
      # Specify a source or an artifact, but not both.
      Sources:
        - source-name-1
      Artifacts:
        - request-payload
      Variables:
        - Name: variable-name-1
          Value: variable-value-1
        - Name: variable-name-2
          Value: variable-value-2
    Environment:
      Name: environment-name
      Connections:
        - Name: account-connection-name
          Role: iam-role-name
    Configuration:
      Function: my-function|function-arn
      AWSRegion: us-west-2
      # Specify RequestPayload or RequestPayloadFile, but not both.
      RequestPayload: '{"firstname": "Li", lastname: "Jean", "company": "ExampleCo",
 "team": "Development"}'
      RequestPayloadFile: my/request-payload.json
      ContinueOnError: true|false
      LogType: Tail|None
      ResponseFilters: '{"name": ".name", "company": ".department.company"}'
    Outputs:
      Artifacts:
        - Name: lambda_artifacts
          Files:
            - "lambda-response.json"
```

LambdaInvoke

(必須)

アクションの名前を指定します。すべてのアクション名は、ワークフロー内で一意である必要があります。アクション名は、英数字 (a～z、A～Z、0～9)、ハイフン (-)、アンダースコア (_) に制限されています。スペースは使用できません。引用符を使用してアクション名で特殊文字やスペースを有効にすることはできません。

デフォルト: `Lambda_Invoke_Action_Workflow_nn`。

対応する UI: 設定タブ/アクション名

Identifier

(*LambdaInvoke*/Identifier)

(必須)

アクションを識別します。バージョンを変更しない限り、このプロパティを変更しないでください。詳細については、「アクションのメジャー、マイナー、またはパッチバージョンの指定」を参照してください。

デフォルト: `aws/lambda-invoke@v1`。

対応する UI: ワークフロー図/LambdaInvoke_nn/aws/lambda-invoke@v1 ラベル

DependsOn

(*LambdaInvoke*/DependsOn)

(オプション)

このアクションを実行するために正常に実行する必要があるアクション、アクショングループ、またはゲートを指定します。

「依存」機能の詳細については、「」を参照してください他のアクションに依存するようにアクションを設定する。

対応する UI: の入力タブ/依存 - オプション

Compute

(*LambdaInvoke*/Compute)

(オプション)

ワークフローアクションを実行するために使用されるコンピューティングエンジン。コンピューティングはワークフローレベルまたはアクションレベルで指定できますが、両方を指定することはできません。ワークフローレベルで指定すると、コンピューティング設定はワークフローで定義されたすべてのアクションに適用されます。ワークフローレベルでは、同じインスタンスで複数のアクションを実行することもできます。詳細については、「アクション間でのコンピューティングの共有」を参照してください。

対応する UI: なし

Type

(*LambdaInvoke*/Compute/Type)

(Computeが含まれている場合は必須)

コンピューティングエンジンのタイプ。次のいずれかの値を使用できます。

- EC2 (ビジュアルエディタ) または EC2 (YAML エディタ)

  アクション実行中の柔軟性のために最適化されました。
- Lambda (ビジュアルエディタ) または Lambda (YAML エディタ)

  アクションの起動速度を最適化しました。

コンピューティングタイプの詳細については、「コンピューティングタイプ」を参照してください。

対応する UI: 設定タブ/コンピューティングタイプ

Fleet

(*LambdaInvoke*/Compute/Fleet)

(オプション)

ワークフローまたはワークフローアクションを実行するマシンまたはフリートを指定します。オンデマンドフリートでは、アクションが開始されると、ワークフローは必要なリソースをプロビジョニングし、アクションが終了するとマシンは破棄されます。オンデマンドフリートの例: `Linux.x86-64.Large`、`Linux.x86-64.XLarge`。オンデマンドフリートの詳細については、「」を参照してくださいオンデマンドフリートのプロパティ。

プロビジョニングされたフリートでは、ワークフローアクションを実行するように一連の専用マシンを設定します。これらのマシンはアイドル状態のままで、すぐにアクションを処理できます。プロビジョニングされたフリートの詳細については、「」を参照してくださいプロビジョニングされたフリートのプロパティ。

`Fleet` を省略した場合、デフォルトは ですLinux.x86-64.Large。

対応する UI: 設定タブ/コンピューティングフリート

Timeout

(*LambdaInvoke*/Timeout)

(必須)

CodeCatalyst アクションが終了するまでに実行できる時間を分単位で指定します (YAML エディタ)、または時間と分単位で指定します (ビジュアルエディタ)。最小値は 5 分で、最大値は「」で説明されていますワークフローのクォータ。デフォルトのタイムアウトは、最大タイムアウトと同じです。

対応する UI: 設定タブ/タイムアウト - オプション

Inputs

(*LambdaInvoke*/Inputs)

(必須)

`Inputs` セクションでは、ワークフローの実行中にAWS Lambda 呼び出しアクションに必要なデータを定義します。

> **Note**
>
> AWS Lambda 呼び出しアクションごとに許可される入力 (ソースまたはアーティファクト) は 1 つだけです。変数はこの合計にはカウントされません。

対応する UI: Inputs タブ

Sources

(*LambdaInvoke*/Inputs/Sources)

(RequestPayloadFileが指定されている場合は必須)

リクエストペイロード JSON ファイルをAWS Lambda 呼び出しアクションに渡し、このペイロードファイルがソースリポジトリに保存されている場合は、そのソースリポジトリのラベルを指定します。現在、サポートされているラベルは のみですWorkflowSource。

リクエストペイロードファイルがソースリポジトリに含まれていない場合は、別のアクションによって生成されたアーティファクトに存在する必要があります。

ペイロードファイルの詳細については、「」を参照してくださいRequestPayloadFile。

> **Note**
>
> ペイロードファイルを指定する代わりに、 RequestPayloadプロパティを使用してペイロードの JSON コードを アクションに直接追加できます。詳細については、「RequestPayload」を参照してください。

sources の詳細については、「ワークフローをソースリポジトリに接続する」を参照してください。

対応する UI: 入力タブ/ソース - オプション

Artifacts - input

(*LambdaInvoke*/Inputs/Artifacts)

(RequestPayloadFileが指定されている場合は必須)

リクエストペイロード JSON ファイルをAWS Lambda 呼び出しアクションに渡し、このペイロードファイルが前のアクションの出力アーティファクトに含まれている場合は、ここでそのアーティファクトを指定します。

ペイロードファイルの詳細については、「」を参照してくださいRequestPayloadFile。

> **Note**
>
> ペイロードファイルを指定する代わりに、 RequestPayloadプロパティを使用してペイロードの JSON コードを アクションに直接追加できます。詳細については、「RequestPayload」を参照してください。

アーティファクトの例などの詳細については、「」を参照してくださいアーティファクトを使用したワークフロー内のアクション間のデータの共有。

対応する UI: 設定タブ/アーティファクト - オプション

Variables - input

(*LambdaInvoke*/Inputs/Variables)

(オプション)

アクションで使用できるようにしたい入力変数を定義する名前と値のペアのシーケンスを指定します。変数名は、英数字 (a～z、A～Z、0～9)、ハイフン (-)、アンダースコア (_) に制限されています。スペースは使用できません。変数名で特殊文字やスペースを有効にするために引用符を使用することはできません。

例を含む変数の詳細については、「」を参照してくださいワークフローでの変数の設定と使用。

対応する UI: 入力タブ/変数 - オプション

Environment

(*LambdaInvoke*/Environment)

(必須)

アクションで使用する CodeCatalyst 環境を指定します。

環境の詳細については、環境を使用して AWS アカウント および VPCsにデプロイする CodeCatalyst「」および「」を参照してください環境を作成する。

対応する UI: 設定タブ/'環境/アカウント/ロール'/環境

Name

(*LambdaInvoke*/Environment/Name)

(Environmentが含まれている場合は必須)

アクションに関連付ける既存の環境の名前を指定します。

対応する UI: 設定タブ/'環境/接続/ロール'/環境

Connections

(*LambdaInvoke*/Environment/Connections)

(Environmentが含まれている場合は必須)

アクションに関連付けるアカウント接続を指定します。で最大 1 つのアカウント接続を指定できますEnvironment。

アカウント接続の詳細については、「」を参照してください接続された AWS リソースへのアクセスを許可する AWS アカウント。アカウント接続を環境に関連付ける方法については、「」を参照してください環境を作成する。

対応する UI: 設定タブ/'環境/接続/ロール'/接続

Name

(*LambdaInvoke*/Environment/Connections/Name)

(必須)

アカウント接続の名前を指定します。

対応する UI: 設定タブ/'環境/接続/ロール'/接続

Role

(*LambdaInvoke*/Environment/Connections/Role)

(必須)

AWS Lambda 呼び出しアクションが Lambda 関数へのアクセス AWS と呼び出しに使用する IAM ロールの名前を指定します。このロールに以下が含まれていることを確認してください。

- 次のアクセス許可ポリシー：

> **Warning**
> アクセス許可を次のポリシーに示すものに制限します。より広範なアクセス許可を持つロールを使用すると、セキュリティ上のリスクが生じる可能性があります。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:aws-region:aws-account:function:function-
name"
        }
    ]
}
```

- 次のカスタム信頼ポリシー：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                 ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

このロールがアカウント接続に関連付けられていることを確認します。IAM ロールとアカウント接続の関連付けの詳細については、「」を参照してくださいアカウント接続への IAM ロールの追加。

> **Note**
>
> 必要に応じて、ここでCodeCatalystWorkflowDevelopmentRole-*spaceName*ロールの名前を指定できます。このロールの詳細については、「アカウントとスペースのCodeCatalystWorkflowDevelopmentRole-***spaceName***ロールの作成」を参照してください。CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールには、セキュリティリスクをもたらす可能性のある非常に広範なアクセス許可があることを理解します。この

ロールは、セキュリティが懸念されないチュートリアルとシナリオでのみ使用することをお勧めします。

対応する UI: 設定タブ/'環境/接続/ロール'/ロール

Configuration

(*LambdaInvoke*/Configuration)

(必須)

アクションの設定プロパティを定義できるセクション。

対応する UI: 設定タブ

Function

(*LambdaInvoke*/Configuration/Function)

(必須)

このアクションが呼び出す AWS Lambda 関数を指定します。関数の名前、またはその Amazon リソースネーム (ARN) を指定できます。名前または ARN は Lambda コンソールで確認できます。

> **Note**
>
> Lambda 関数が存在する AWS アカウントは、 で指定されたアカウントとは異なる場合がありますConnections:。

対応する UI: 設定タブ/機能

AWSRegion

(*LambdaInvoke*/Configuration/AWSRegion)

(必須)

AWS Lambda 関数が存在する AWS リージョンを指定します。リージョンコードのリストについては、「」の「リージョンエンドポイント」を参照してくださいAWS 全般のリファレンス。

対応する UI: 設定タブ/送信先バケット - オプション

RequestPayload

(*LambdaInvoke*/Configuration/RequestPayload)

(オプション)

AWS Lambda 呼び出しアクションにリクエストペイロードを渡す場合は、ここでリクエストペイロードを JSON 形式で指定します。

リクエストペイロードの例：

```
'{ "key": "value" }'
```

Lambda 関数にリクエストペイロードを渡さない場合は、このプロパティを省略します。

> **Note**
>
> `RequestPayload` または `RequestPayloadFile` を指定できます。両方を指定することはできません。

リクエストペイロードの詳細については、AWS Lambda 「 API リファレンス」の「呼び出し」トピックを参照してください。

対応する UI: 設定タブ/リクエストペイロード - オプション

RequestPayloadFile

(*LambdaInvoke*/Configuration/RequestPayloadFile)

(オプション)

AWS Lambda 呼び出しアクションにリクエストペイロードを渡す場合は、ここでこのリクエストペイロードファイルへのパスを指定します。ファイルは JSON 形式である必要があります。

リクエストペイロードファイルは、ソースリポジトリまたは前のアクションのアーティファクトに存在することができます。ファイルパスは、ソースリポジトリまたはアーティファクトルートを基準にしています。

Lambda 関数にリクエストペイロードを渡さない場合は、このプロパティを省略します。

> **Note**
>
> RequestPayload または RequestPayloadFile を指定できます。両方を指定することはできません。

リクエストペイロードファイルの詳細については、AWS Lambda 「 API リファレンス」の「呼び出し」トピックを参照してください。

対応する UI: 設定タブ/リクエストペイロードファイル - オプション

ContinueOnError

(*LambdaInvoke*/Configuration/RequestPayloadFile)

(オプション)

AWS Lambda 呼び出しされた AWS Lambda 関数が失敗した場合でも、呼び出しアクションを成功としてマークするかどうかを指定します。Lambda trueに障害が発生してもワークフロー内の後続のアクションを開始できるように、このプロパティを に設定することを検討してください。

デフォルトでは、Lambda 関数が失敗した場合 (ビジュアルエディタまたは YAML エディタでは「オフ」) 、 アクションfalseは失敗します。

対応する UI: 設定タブ/エラーの続行

LogType

(*LambdaInvoke*/Configuration/LogType)

(オプション)

呼び出し後に Lambda 関数からのレスポンスにエラーログを含めるかどうかを指定します。これらのログは、 CodeCatalyst コンソールの Lambda 呼び出しアクションのログタブで表示できます。可能な値は以下のとおりです。

- Tail – ログを返す
- None – ログを返さない

デフォルトはテール です。

ログタイプの詳細については、AWS Lambda 「 API リファレンス」の「呼び出し」トピックを参照してください。

ログの表示の詳細については、「ワークフローの実行ステータスと詳細の表示」を参照してください。

対応する UI: 設定タブ/ログタイプ

ResponseFilters

(*LambdaInvoke*/Configuration/ResponseFilters)

(オプション)

出力変数に変換する Lambda レスポンスペイロードのキーを指定します。その後、ワークフロー内の後続のアクションで出力変数を参照できます。の変数の詳細については、 CodeCatalyst「」を参照してくださいワークフローでの変数の設定と使用。

例えば、レスポンスペイロードは次のようになります。

```
responsePayload = {
  "name": "Saanvi",
  "location": "Seattle",
  "department": {
    "company": "Amazon",
    "team": "AWS"
  }
}
```

レスポンスフィルターは次のようになります。

```
Configuration:
  ...
  ResponseFilters: '{"name": ".name", "company": ".department.company"}'
```

アクションは、次の出力変数を生成します。

キー	値
name	Saanvi
company	Amazon

その後、後続のアクションで nameおよび company変数を参照できます。

でキーを指定しない場合ResponseFilters、 アクションは Lambda レスポンスの各トップレベルキーを出力変数に変換します。詳細については、「AWS Lambda 「呼び出し」アクションによって生成される変数 」を参照してください。

レスポンスフィルターを使用して、生成された出力変数を実際に使用する変数のみに制限することを検討してください。

対応する UI: 設定タブ/レスポンスフィルター - オプション

Outputs

(*LambdaInvoke*/Outputs)

(オプション)

ワークフローの実行中に アクションによって出力されるデータを定義します。

対応する UI: 出力タブ

Artifacts

(*LambdaInvoke*/Outputs/Artifacts)

(オプション)

アクションによって生成されたアーティファクトを指定します。これらのアーティファクトは、他のアクションで入力として参照できます。

アーティファクトの例などの詳細については、「」を参照してくださいアーティファクトを使用したワークフロー内のアクション間のデータの共有。

対応する UI: タブ/アーティファクト/ビルドアーティファクト名を出力する

Name

(*LambdaInvoke*/Outputs/Artifacts/Name)

(オプション)

Lambda 関数によって返される Lambda レスポンスペイロードを含むアーティファクトの名前を指定します。デフォルト値は、lambda_artifactsです。アーティファクトを指定しない場

合、Lambda レスポンスペイロードをアクションのログで表示できます。これは、 CodeCatalyst コンソールのアクションのログタブで確認できます。ログの表示の詳細については、「ワークフローの実行ステータスと詳細の表示」を参照してください。

対応する UI: 出力タブ/アーティファクト/ビルドアーティファクト名

Files

(*LambdaInvoke*/Outputs/Artifacts/Files)

(オプション)

アーティファクトに含めるファイルを指定します。Lambda レスポンスペイロードファイルが含まれる`lambda-response.json`ように を指定する必要があります。

対応する UI: ビルドによって生成されたタブ/アーティファクト/ファイルを出力する

ワークフローを使用した Amazon ECS タスク定義ファイルの変更

このセクションでは、 CodeCatalyst ワークフローを使用して Amazon Elastic Container Service (Amazon ECS) タスク定義ファイルの `image`フィールドを更新する方法について説明します。これを行うには、ワークフローに Amazon ECS タスク定義アクションのレンダリングを追加する必要があります。このアクションは、タスク定義ファイルのイメージフィールドを、実行時にワークフローによって指定された Docker イメージ名で更新します。

> **Note**
>
> このアクションを使用して、タスク定義の `environment`フィールドを環境変数で更新することもできます。

このアクションを使用するタイミング

これは、コミット ID やタイムスタンプなどの動的コンテンツを使用して Docker イメージを構築してタグ付けするワークフローがある場合に使用します。

タスク定義ファイルに常に同じままのイメージ値が含まれている場合は、このアクションを使用しないでください。この場合、イメージの名前をタスク定義ファイルに手動で入力できます。

「Amazon ECS タスク定義のレンダリング」アクションの仕組み

ワークフローのビルドおよび Amazon ECS アクションへのデプロイで、Amazon ECS タスク定義アクションのレンダリングを使用する必要があります。 これらのアクションは、まとめて次のように機能します。

1. ビルドアクションは Docker イメージをビルドし、名前、コミット ID、タイムスタンプ、またはその他の動的コンテンツでタグ付けします。例えば、ビルドアクションは次のようになります。

```
MyECSWorkflow
  Actions:
    BuildAction:
      Identifier: aws/build@v1
      ...
      Configuration:
        Steps:
        # Build, tag, and push the Docker image...
          - Run: docker build -t MyDockerImage:${WorkflowSource.CommitId} .
          ...
```

   前述のコードでは、 `docker build -t`ディレクティブは が Docker イメージを構築し、アクションの実行時にコミット ID でタグ付けするように指示しています。生成されたイメージ名は次のようになります。

   `MyDockerImage:a37bd7e`

2. Amazon ECS タスク定義のレンダリングアクションは、動的に生成されたイメージ名 `MyDockerImage:a37bd7e`を、次のようにタスク定義ファイルに追加します。

```
{
    "executionRoleArn": "arn:aws:iam::account_ID:role/codecatalyst-ecs-task-
execution-role",
    "containerDefinitions": [
        {
            "name": "codecatalyst-ecs-container",
            "image":  MyDockerImage:a37bd7e,
            "essential": true,
            ...
            "portMappings": [
                {
                    "hostPort": 80,
                    "protocol": "tcp",
```

```
                    "containerPort": 80
                }
            ]
        }
    ],
...
}
```

オプションで、次のように Amazon ECS タスク定義アクションをレンダリングして、タスク定義に環境変数を追加することもできます。

```
{
  "executionRoleArn": "arn:aws:iam::account_ID:role/codecatalyst-ecs-task-execution-role",
  "containerDefinitions": [
    {
      "name": "codecatalyst-ecs-container",
      "image":  MyDockerImage:a37bd7e,
      ...
      "environment": [
        {
          name": "ECS_LOGLEVEL",
          value": "info"
        }
      ]
    }
  ],
...
}
```

環境変数の詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「環境変数の指定」を参照してください。

3. Amazon ECS にデプロイ アクションは、更新されたタスク定義ファイルを Amazon ECS に登録します。更新されたタスク定義ファイルを登録すると、新しいイメージが Amazon ECS `MyDockerImage:a37bd7e`にデプロイされます。

トピック

- Amazon ECS タスク定義ファイルを変更するワークフローの例
- 「Amazon ECS タスク定義のレンダリング」アクションの追加

- 更新されたタスク定義ファイルの表示
- 「Amazon ECS タスク定義のレンダリング」アクションによって生成される変数
- 「Amazon ECS タスク定義のレンダリング」アクション YAML 定義リファレンス

Amazon ECS タスク定義ファイルを変更するワークフローの例

以下は、Amazon ECS タスク定義アクションのレンダリングと、ビルドおよびデプロイアクションを含むワークフロー全体の例です。ワークフローの目的は、Docker イメージを構築して Amazon ECS クラスターにデプロイすることです。ワークフローは、順番に実行される次の構成要素で構成されます。

- トリガー — このトリガーは、変更をソースリポジトリにプッシュすると、ワークフローの実行を自動的に開始します。トリガーについての詳細は、「トリガーを使用したワークフローの自動実行の開始」を参照してください。
- ビルドアクション (`BuildDocker`) — トリガー時に、 アクションは Dockerfile を使用して Docker イメージをビルドし、コミット ID でタグ付けして、そのイメージを Amazon ECR にプッシュします。ビルドアクションの詳細については、「」を参照してくださいワークフローによる構築。
- Amazon ECS タスク定義アクションのレンダリング (`RenderTaskDef`) – ビルドアクションが完了すると、このアクションは、ソースリポジトリのルート`taskdef.json`にある既存の を、正しいコミット ID を含む`image`フィールド値で更新します。更新されたファイルを新しいファイル名 (`task-definition-`*`random-string`*`.json`) で保存し、このファイルを含む出力アーティファクトを作成します。レンダリングアクションは、 という変数も生成`task-definition`し、新しいタスク定義ファイルの名前に設定します。アーティファクトと変数はデプロイアクションで使用されます。次に です。
- Amazon ECS へのデプロイアクション (`DeployToECS`) – Amazon ECS タスク定義アクションのレンダリングが完了すると、Amazon ECS へのデプロイアクションは、レンダーアクション (`TaskDefArtifact`) によって生成された出力アーティファクトを検索し、その中に`task-definition-`*`random-string`*`.json`ファイルを見つけて、Amazon ECS サービスに登録します。次に、Amazon ECS サービスは `task-definition-`*`random-string`*`.json` ファイルの指示に従って、Amazon ECS クラスター内で Amazon ECS タスクと関連する Docker イメージコンテナを実行します。

```
Name: codecatalyst-ecs-workflow
SchemaVersion: 1.0
```

```
Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  BuildDocker:
    Identifier: aws/build@v1
    Environment:
      Name: codecatalyst-ecs-environment
      Connections:
        - Name: codecatalyst-account-connection
          Role: codecatalyst-ecs-build-role
    Inputs:
      Variables:
        - Name: REPOSITORY_URI
          Value: 111122223333.dkr.ecr.us-east-2.amazonaws.com/codecatalyst-ecs-image-
repo
        - Name: IMAGE_TAG
          Value: ${WorkflowSource.CommitId}
    Configuration:
      Steps:
        #pre_build:
        - Run: echo Logging in to Amazon ECR...
        - Run: aws --version
        - Run: aws ecr get-login-password --region us-east-2 | docker login --username
 AWS --password-stdin 111122223333.dkr.ecr.us-east-2.amazonaws.com
        #build:
        - Run: echo Build started on `date`
        - Run: echo Building the Docker image...
        - Run: docker build -t $REPOSITORY_URI:latest .
        - Run: docker tag $REPOSITORY_URI:latest $REPOSITORY_URI:$IMAGE_TAG
        #post_build:
        - Run: echo Build completed on `date`
        - Run: echo Pushing the Docker images...
        - Run: docker push $REPOSITORY_URI:latest
        - Run: docker push $REPOSITORY_URI:$IMAGE_TAG

  RenderTaskDef:
    DependsOn:
      - BuildDocker
    Identifier: aws/ecs-render-task-definition@v1
    Inputs:
      Variables:
        - Name: REPOSITORY_URI
```

```
          Value: 111122223333.dkr.ecr.us-east-2.amazonaws.com/codecatalyst-ecs-image-
repo
        - Name: IMAGE_TAG
          Value: ${WorkflowSource.CommitId}
    Configuration:
      task-definition: taskdef.json
      container-definition-name: codecatalyst-ecs-container
      image: $REPOSITORY_URI:$IMAGE_TAG
    # The output artifact contains the updated task definition file.
    # The new file is prefixed with 'task-definition'.
    # The output variable is set to the name of the updated task definition file.
    Outputs:
      Artifacts:
        - Name: TaskDefArtifact
          Files:
            - "task-definition*"
      Variables:
        - task-definition

  DeployToECS:
    Identifier: aws/ecs-deploy@v1
    Environment:
      Name: codecatalyst-ecs-environment
      Connections:
        - Name: codecatalyst-account-connection
          Role: codecatalyst-ecs-deploy-role
    #Input artifact contains the updated task definition file.
    Inputs:
      Sources: []
      Artifacts:
        - TaskDefArtifact
    Configuration:
      region: us-east-2
      cluster: codecatalyst-ecs-cluster
      service: codecatalyst-ecs-service
      task-definition: ${RenderTaskDef.task-definition}
```

「Amazon ECS タスク定義のレンダリング」アクションの追加

次の手順を使用して、ワークフローに Amazon ECS タスク定義アクションをレンダリングします。

前提条件

開始する前に、Docker イメージを動的に生成するビルドアクションを含むワークフローがあることを確認してください。詳細については、前述のワークフローの例を参照してください。

Visual

ビジュアルエディタを使用して「Amazon ECS タスク定義のレンダリング」アクションを追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. ビジュアル を選択します。
7. 左上で、+ Actions を選択してアクションカタログを開きます。
8. ドロップダウンリストから Amazon CodeCatalystを選択します。
9. Render Amazon ECS タスク定義アクションを検索し、次のいずれかを実行します。

   - プラス記号 (+) を選択してワークフロー図にアクションを追加し、設定ペインを開きます。

     または

   - Amazon ECS タスク定義 のレンダリングを選択します。アクションの詳細ダイアログボックスが表示されます。このダイアログボックスで、次の操作を行います。
     - (オプション) ソースを表示 を選択して、アクションのソースコード を表示します。
     - ワークフローに追加 を選択して、ワークフロー図にアクションを追加し、その設定ペインを開きます。
10. 入力タブと設定タブで、必要に応じてフィールドに入力します。各フィールドの説明については、「」を参照してください「Amazon ECS タスク定義のレンダリング」アクション YAML 定義リファレンス。このリファレンスでは、YAML エディタとビジュアルエディタの両方に表示される各フィールド (および対応する YAML プロパティ値) に関する詳細情報を提供します。
11. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。

12. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

YAML

YAML エディタを使用して「Amazon ECS タスク定義のレンダリング」アクションを追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. YAML を選択します。
7. 左上で + Actions を選択してアクションカタログを開きます。
8. ドロップダウンリストから Amazon CodeCatalystを選択します。
9. Render Amazon ECS タスク定義アクションを検索し、次のいずれかを実行します。
   - プラス記号 (+) を選択してワークフロー図にアクションを追加し、その設定ペインを開きます。

     または

   - Amazon ECS タスク定義 のレンダリングを選択します。アクションの詳細ダイアログボックスが表示されます。このダイアログボックスで、次の操作を行います。
     - (オプション) ソースを表示 を選択して、アクションのソースコード を表示します。
     - ワークフローに追加 を選択して、ワークフロー図にアクションを追加し、その設定ペインを開きます。
10. 必要に応じて YAML コードのプロパティを変更します。使用可能な各プロパティの説明は、「」に記載されています「Amazon ECS タスク定義のレンダリング」アクション YAML 定義リファレンス。
11. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
12. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

次のステップ

レンダーアクションを追加したら、「」の手順に従って、Amazon ECS へのデプロイアクションをワークフローに追加しますワークフローを使用した Amazon Elastic Container Service (ECS) へのアプリケーションのデプロイ。デプロイアクションを追加するときに、次の操作を行います。

1. デプロイアクションの入力タブのアーティファクト - オプション で、レンダーアクションによって生成されたアーティファクトを選択します。更新されたタスク定義ファイルが含まれています。

   アーティファクトの詳細については、「アーティファクトを使用したワークフロー内のアクション間のデータの共有」を参照してください。
2. デプロイアクションの設定タブのタスク定義フィールドで、次のアクション変数を指定します。${*action-name*.task-definition}ここで、*action-name* はレンダーアクションの名前ですRenderTaskDef。例えば、。レンダリングアクションは、この変数をタスク定義ファイルの新しい名前に設定します。

   変数の詳細については、「」を参照してくださいワークフローでの変数の設定と使用。

   デプロイアクションの設定方法の詳細については、前述のワークフロー の例を参照してください。

更新されたタスク定義ファイルの表示

更新されたタスク定義ファイルの名前と内容を表示できます。

更新されたタスク定義ファイルの名前を表示するには、Amazon ECS タスク定義のレンダリングアクションで処理された後。

1. 完了したレンダーアクションを含む実行を検索します。

   a. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。

   b. プロジェクトを選択します。

   c. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。

   d. レンダリングアクションを含むワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。

   e. 完了したレンダリングアクションを含む実行を選択します。

2. ワークフロー図で、レンダリングアクションを選択します。
3. 出力 を選択します。
4. 変数 を選択します。
5. タスク定義ファイル名が表示されます。のようになりますtask-definition--259-0a2r7gxlTF5X-.json。

更新されたタスク定義ファイルの内容を表示するには

1. 完了したレンダーアクションを含む実行を検索します。
   a. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
   b. プロジェクトを選択します。
   c. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
   d. レンダリングアクションを含むワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
   e. 完了したレンダリングアクションを含む実行を選択します。
2. ワークフロー実行で、ビジュアル と YAML の横にある上部で、ワークフロー出力 を選択します。
3. アーティファクト セクションで、更新されたタスク定義ファイルを含むアーティファクトの横にあるダウンロード を選択します。このアーティファクトには、Produced by 列がレンダリングアクションの名前に設定されます。
4. .zip ファイルを開き、タスク定義 .json ファイルを表示します。

「Amazon ECS タスク定義のレンダリング」アクションによって生成される変数

Render Amazon ECS タスク定義アクションは、実行時に以下の変数を生成して設定します。これらは事前定義された変数 と呼ばれます。

ワークフローでこれらの変数を参照する方法については、「」を参照してください事前定義された変数の使用。

キー	値
タスク定義	Render Amazon ECS タスク定義アクションによって更新されたタスク定義ファイルに付けられた名前。バケット名は `task-definition- random-string .json` 形式に従います。 例: `task-definition--259-0a2r7gxlTF5Xr.json`

「Amazon ECS タスク定義のレンダリング」アクション YAML 定義リファレンス

以下は、Amazon ECS タスク定義アクションのレンダリングの YAML 定義です。このアクションの使用方法については、「」を参照してくださいワークフローを使用した Amazon ECS タスク定義ファイルの変更。

このアクション定義は、より広範なワークフロー定義ファイル内のセクションとして存在します。ファイルの詳細については、「ワークフロー YAML 定義」を参照してください。

> **Note**
>
> 後続の YAML プロパティのほとんどには、対応する UI 要素がビジュアルエディタにあります。UI 要素を検索するには、Ctrl+F を使用します。 要素は、関連付けられた YAML プロパティとともに一覧表示されます。

```
# The workflow definition starts here.
# See ######### for details.

Name: MyWorkflow
SchemaVersion: 1.0
Actions:

# The action definition starts here.
  ECSRenderTaskDefinition_nn:
```

```
    Identifier: aws/ecs-render-task-definition@v1
    DependsOn:
      - build-action
    Compute:
      Type: EC2 | Lambda
      Fleet: fleet-name
    Timeout: timeout-minutes
    Inputs:
      # Specify a source or an artifact, but not both.
      Sources:
        - source-name-1
      Artifacts:
        - task-definition-artifact
      Variables:
        - Name: variable-name-1
          Value: variable-value-1
        - Name: variable-name-2
          Value: variable-value-2
    Configuration
      task-definition: task-definition-path
      container-definition-name: container-definition-name
      image: docker-image-name
      environment-variables:
        - variable-name-1=variable-value-1
        - variable-name-2=variable-value-2
    Outputs:
      Artifacts:
        - Name: TaskDefArtifact
          Files: "task-definition*"
      Variables:
        - task-definition
```

ECSRenderTaskDefinition

(必須)

アクションの名前を指定します。すべてのアクション名は、ワークフロー内で一意である必要があります。アクション名は、英数字 (a～z、A～Z、0～9)、ハイフン (-)、アンダースコア (_) に制限されています。スペースは使用できません。引用符を使用してアクション名で特殊文字やスペースを有効にすることはできません。

デフォルト: ECSRenderTaskDefinition_nn。

対応する UI: 設定タブ/アクション名

Identifier

(*ECSRenderTaskDefinition*/Identifier)

(必須)

アクションを識別します。バージョンを変更しない限り、このプロパティを変更しないでください。詳細については、「アクションのメジャー、マイナー、またはパッチバージョンの指定」を参照してください。

デフォルト: `aws/ecs-render-task-definition@v1`。

対応する UI: ワークフロー図/ECSRenderTaskDefinition_nn/aws/ecs-render-task-definition@v1 ラベル

DependsOn

(*ECSRenderTaskDefinition*/DependsOn)

(オプション)

このアクションを実行するために正常に実行する必要があるアクション、アクショングループ、またはゲートを指定します。

「依存」機能の詳細については、「」を参照してください他のアクションに依存するようにアクションを設定する。

対応する UI: の入力タブ/依存 - オプション

Compute

(*ECSRenderTaskDefinition*/Compute)

(オプション)

ワークフローアクションを実行するために使用されるコンピューティングエンジン。コンピューティングはワークフローレベルまたはアクションレベルで指定できますが、両方を指定することはできません。ワークフローレベルで指定すると、コンピューティング設定はワークフローで定義されたすべてのアクションに適用されます。ワークフローレベルでは、同じインスタンスで複数のアクションを

実行することもできます。詳細については、「アクション間でのコンピューティングの共有」を参照してください。

対応する UI: なし

Type

(*ECSRenderTaskDefinition*/Compute/Type)

(Computeが含まれている場合は必須)

コンピューティングエンジンのタイプ。次のいずれかの値を使用できます。

- EC2 (ビジュアルエディタ) または EC2 (YAML エディタ)

  アクション実行中の柔軟性のために最適化されました。
- Lambda (ビジュアルエディタ) または Lambda (YAML エディタ)

  アクションの起動速度を最適化しました。

コンピューティングタイプの詳細については、「コンピューティングタイプ」を参照してください。

対応する UI: 設定タブ/コンピューティングタイプ

Fleet

(*ECSRenderTaskDefinition*/Compute/Fleet)

(オプション)

ワークフローまたはワークフローアクションを実行するマシンまたはフリートを指定します。オンデマンドフリートでは、アクションが開始されると、ワークフローは必要なリソースをプロビジョニングし、アクションが終了するとマシンは破棄されます。オンデマンドフリートの例: `Linux.x86-64.Large`、`Linux.x86-64.XLarge`。オンデマンドフリートの詳細については、「」を参照してくださいオンデマンドフリートのプロパティ。

プロビジョニングされたフリートでは、ワークフローアクションを実行するように一連の専用マシンを設定します。これらのマシンはアイドル状態のままで、すぐにアクションを処理できます。プロビジョニングされたフリートの詳細については、「」を参照してくださいプロビジョニングされたフリートのプロパティ。

`Fleet` を省略した場合、デフォルトは ですLinux.x86-64.Large。

対応する UI: 設定タブ/コンピューティングフリート

Timeout

(*ECSRenderTaskDefinition*/Timeout)

(オプション)

CodeCatalyst アクションが終了するまでに実行できる時間を分単位で指定します (YAML エディタ)、または時間と分単位で指定します (ビジュアルエディタ)。最小値は 5 分で、最大値は「」で説明されていますワークフローのクォータ。デフォルトのタイムアウトは、最大タイムアウトと同じです。

対応する UI: 設定タブ/タイムアウト - オプション

Inputs

(*ECSRenderTaskDefinition*/Inputs)

(オプション)

`Inputs` セクションでは、ワークフローの実行中に がECSRenderTaskDefinition必要とするデータを定義します。

> **Note**
>
> Render Amazon ECS タスク定義アクションごとに許可される入力 (ソースまたはアーティファクト) は 1 つだけです。変数はこの合計にはカウントされません。

対応する UI: Inputs タブ

Sources

(*ECSRenderTaskDefinition*/Inputs/Sources)

(タスク定義ファイルがソースリポジトリに保存されている場合に必須)

タスク定義ファイルがソースリポジトリに保存されている場合は、そのソースリポジトリのラベルを指定します。現在、サポートされているラベルは のみですWorkflowSource。

タスク定義ファイルがソースリポジトリに含まれていない場合は、別のアクションによって生成されたアーティファクトに存在する必要があります。

sources の詳細については、「ワークフローをソースリポジトリに接続する」を参照してください。

対応する UI: 入力タブ/ソース - オプション

Artifacts - input

(*ECSRenderTaskDefinition*/Inputs/Artifacts)

(タスク定義ファイルが前のアクションの出力アーティファクトに保存されている場合に必要です)

デプロイするタスク定義ファイルが、前のアクションによって生成されたアーティファクトに含まれている場合は、ここでそのアーティファクトを指定します。タスク定義ファイルがアーティファクトに含まれていない場合は、ソースリポジトリに存在する必要があります。

アーティファクトの例などの詳細については、「」を参照してくださいアーティファクトを使用したワークフロー内のアクション間のデータの共有。

対応する UI: 設定タブ/アーティファクト - オプション

Variables - input

(*ECSRenderTaskDefinition*/Inputs/Variables)

(必須)

アクションで使用できるようにしたい入力変数を定義する名前と値のペアのシーケンスを指定します。変数名は、英数字 (a ~ z、A ~ Z、0 ~ 9)、ハイフン (-)、アンダースコア (_) に制限されています。スペースは使用できません。変数名で特殊文字やスペースを有効にするために引用符を使用することはできません。

例を含む変数の詳細については、「」を参照してくださいワークフローでの変数の設定と使用。

対応する UI: 入力タブ/変数 - オプション

Configuration

(*ECSRenderTaskDefinition*/Configuration)

(必須)

アクションの設定プロパティを定義できるセクション。

対応する UI: 設定タブ

task-definition

(*ECSRenderTaskDefinition*/Configuration/task-definition)

(必須)

既存のタスク定義ファイルへのパスを指定します。ファイルがソースリポジトリにある場合、パスはソースリポジトリのルートフォルダを基準にしています。ファイルが以前のワークフローアクションのアーティファクトに存在する場合、パスはアーティファクトルートフォルダを基準にしています。タスク定義ファイルの詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「タスク定義」を参照してください。

対応する UI: 設定タブ/タスク定義

container-definition-name

(*ECSRenderTaskDefinition*/Configuration/container-definition-name)

(必須)

Docker イメージを実行するコンテナの名前を指定します。この名前は`containerDefinitions`、タスク定義ファイルの 、 `name`フィールドにあります。詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「名前」を参照してください。

対応する UI: 設定タブ/コンテナ名

image

(*ECSRenderTaskDefinition*/Configuration/image)

(必須)

Render Amazon ECS タスク定義アクションでタスク定義ファイルに追加する Docker イメージの名前を指定します。アクションは`containerDefinitions`、タスク定義ファイルの 、 `image`フィー

ルドにこの名前を追加します。値が imageフィールドにすでに存在する場合、アクションによって上書きされます。イメージ名に変数を含めることができます。

例:

を指定するとMyDockerImage:${WorkflowSource.CommitId}、 アクションはタスク定義ファイルMyDockerImage:*commit-id*に追加されます。*### commit-id* は、ワークフローによって実行時に生成されるコミット ID です。

を指定するとmy-ecr-repo/image-repo:$(date +%m-%d-%y-%H-%m-%s)、アクションは *my-ecr-repo*/image-repo:*date +%m-%d-%y-%H-%m-%s* をタスク定義ファイルに追加します。ここで、 *my-ecr-repo* は Amazon Elastic Container Registry (ECR) の URI であり、*date +%m-%d-%y-%H-%m-%s* はワークフローの実行時にmonth-day-year-hour-minute-second生成される形式のタイムスタンプです。

image フィールドの詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「イメージ」を参照してください。変数の詳細については、「」を参照してくださいワークフローでの変数の設定と使用。

対応する UI: 設定タブ/イメージ名

environment-variables

(*ECSRenderTaskDefinition*/Configuration/environment-variables)

(必須)

Render Amazon ECS タスク定義アクションでタスク定義ファイルに追加する環境変数を指定します。アクションは、タスク定義ファイルの containerDefinitions、 environmentフィールドに変数を追加します。変数が ファイルにすでに存在する場合、 アクションは既存の変数の値を上書きし、新しい変数を追加します。Amazon ECS 環境変数の詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「環境変数の指定」を参照してください。

対応する UI: 設定タブ/環境変数 - オプション

Outputs

(*ECSRenderTaskDefinition*/Outputs)

(必須)

ワークフローの実行中に アクションによって出力されるデータを定義します。

対応する UI: 出力タブ

Artifacts

(*ECSRenderTaskDefinition*/Outputs/Artifacts)

(必須)

アクションによって生成されたアーティファクトを指定します。これらのアーティファクトは、他のアクションで入力として参照できます。

アーティファクトの例などの詳細については、「」を参照してくださいアーティファクトを使用したワークフロー内のアクション間のデータの共有。

対応する UI: タブ/アーティファクトを出力する

Name

(*ECSRenderTaskDefinition*/Outputs/Artifacts/Name)

(必須)

更新されたタスク定義ファイルを含むアーティファクトの名前を指定します。デフォルト値は、`MyTaskDefinitionArtifact`です。次に、このアーティファクトを Amazon ECS へのデプロイアクションへの入力として指定する必要があります。このアーティファクトを Amazon ECS へのデプロイアクションへの入力として追加する方法については、「」を参照してくださいAmazon ECS タスク定義ファイルを変更するワークフローの例。

対応する UI: 出力タブ/アーティファクト/名前

Files

(*ECSRenderTaskDefinition*/Outputs/Artifacts/Files)

(必須)

アーティファクトに含めるファイルを指定します。で始まる更新されたタスク定義ファイルが含まれる`task-definition-*`ように `task-definition-`を指定する必要があります。

対応する UI: タブ/アーティファクト/ファイル出力

Variables

(*ECSRenderTaskDefinition*/Outputs/Variables)

(必須)

レンダリングアクションによって設定する変数の名前を指定します。レンダリングアクションは、この変数の値を更新されたタスク定義ファイルの名前 (など) に設定しますtask-definition-*random-string*.json。次に、Amazon ECS にデプロイアクションのタスク定義 (ビジュアルエディタ) または task-definition (yaml エディタ) プロパティでこの変数を指定する必要があります。この変数を Amazon ECS へのデプロイアクションに追加する方法については、Amazon ECS タスク定義ファイルを変更するワークフローの例「」を参照してください。

デフォルト: task-definition

対応する UI: Outputs tab/Variables/Name フィールド

ワークフローでの変数の設定と使用

変数は、 CodeCatalyst ワークフローで参照できる情報を含むキーと値のペアです。

ワークフローで使用できる変数には、次の 2 種類があります。

- ユーザー定義変数 – これらは、ユーザーが定義するキーと値のペアです。
- 事前定義された変数 – これらは、ワークフローによって自動的に出力されるキーと値のペアです。これらを定義する必要はありません。

> **Note**
>
> CodeCatalyst は、変数のように動作し、他のアクションで参照できるGitHub 出力パラメータ もサポートしています。詳細については、「他のアクションで使用できるように GitHub 出力パラメータをエクスポートする」および「 GitHub 出力パラメータの参照」を参照してください。

トピック

- ユーザー定義変数の使用
- 事前定義された変数の使用
- 事前定義された変数のリスト

ユーザー定義変数の使用

ユーザー定義変数は、定義するキーと値のペアです。2 つのタイプがあります。

- プレーンテキスト変数 、または単に変数 - これらは、ワークフロー定義ファイル内でプレーンテキストで定義するキーと値のペアです。
- シークレット — これらは、Amazon CodeCatalyst コンソールの別のシークレットページで定義するキーと値のペアです。キー (名前) はパブリックラベルであり、値にはプライベートに保持する情報が含まれます。キーはワークフロー定義ファイルでのみ指定します。ワークフロー定義ファイル内のパスワードやその他の機密情報の代わりにシークレットを使用します。

> **Note**
>
> 簡潔にするために、このガイドでは 変数 という用語を使用してプレーンテキスト変数 を意味します。

トピック

変数の定義

変数は、次の 2 つの方法で定義できます。

- ワークフローアクションの `Inputs`セクションで — 「入力」セクションの「変数を定義するには」を参照してください。
- ワークフローアクションの `Steps`セクションで — 「ステップ」セクションの「変数を定義するには」を参照してください。

> **Note**
>
> Steps メソッドは、 CodeCatalyst ビルド、テスト、およびGitHub アクションアクションでのみ機能します。これらは、 Steps セクションを含む唯一のアクションであるためです。

例については、「ユーザー定義変数の例」を参照してください。

Visual

「入力」セクションで変数を定義するには (ビジュアルエディタ)

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. ビジュアル を選択します。
7. ワークフロー図で、変数を設定するアクションを選択します。
8. 入力 を選択します。
9. 変数 - オプション で、変数 を追加 を選択し、次の操作を行います。

   アクションで使用できるようにしたい入力変数を定義する名前と値のペアのシーケンスを指定します。変数名は、英数字 (a～z、A～Z、0～9)、ハイフン (-)、アンダースコア (_) に制限されています。スペースは使用できません。変数名で特殊文字やスペースを有効にするために引用符を使用することはできません。

   例を含む変数の詳細については、「」を参照してくださいワークフローでの変数の設定と使用。
10. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
11. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

YAML

「入力」セクションで変数を定義するには (YAML エディタ)

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. YAML を選択します。
7. ワークフローアクションで、次のようなコードを追加します。

```
action-name:
  Inputs:
    Variables:
      - Name: variable-name
        Value: variable-value
```

その他の例については、「ユーザー定義変数の例」を参照してください。詳細については、アクションのワークフロー YAML 定義「」を参照してください。

8. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
9. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

Visual

「ステップ」セクションで変数を定義するには (ビジュアルエディタ)

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。

5. [編集] を選択します。
6. ビジュアル を選択します。
7. ワークフロー図で、変数を設定するアクションを選択します。
8. [設定] を選択します。
9. Shell コマンドまたは GitHub Actions YAML のいずれか使用可能な方で、アクションの に変数をSteps明示的または暗黙的に定義します。

   - 変数を明示的に定義するには、 Stepsセクションに直接 bash コマンドに含めます。
   - 変数を暗黙的に定義するには、アクションの Stepsセクションで参照されているファイルで指定します。

   例については、「ユーザー定義変数の例」を参照してください。詳細については、 アクションのワークフロー YAML 定義「」を参照してください。
10. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
11. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

YAML

「ステップ」セクションで変数を定義するには (YAML エディタ)

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. YAML を選択します。
7. ワークフローアクションで、明示的または暗黙的にアクションの Stepsセクションに変数を定義します。

   - 変数を明示的に定義するには、 Stepsセクションに直接 bash コマンドに含めます。
   - 変数を暗黙的に定義するには、アクションの Stepsセクションで参照されているファイルで指定します。

例については、「ユーザー定義変数の例」を参照してください。詳細については、 アクションのワークフロー YAML 定義「」を参照してください。

8. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
9. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

シークレットの定義

シークレットは、コンソールのシークレットページで定義します CodeCatalyst。詳細については、「ワークフローでのシークレットの設定と使用」を参照してください。

例えば、次のようなシークレットを定義できます。

- 名前 (キー): **my-password**
- 値: **^*H3#!b9**

シークレットが定義されたら、ワークフロー定義ファイルでシークレットのキー (**my-password**) を指定できます。これを行う方法の例については、「例: シークレットの参照」を参照してください。

他のアクションで使用できるように変数をエクスポートする

次の手順に従って、 アクションから変数をエクスポートし、他のアクションで参照できるようにします。

変数をエクスポートする前に、次の点に注意してください。

- 変数が定義されているアクション内でのみ変数を参照する必要がある場合は、変数をエクスポートする必要はありません。
- すべてのアクションが変数のエクスポートをサポートしているわけではありません。アクションがこの機能をサポートしているかどうかを確認するには、以下のビジュアルエディタの手順を実行して、アクションに出力タブの可変ボタンが含まれているかどうかを確認します。「はい」の場合、変数のエクスポートがサポートされます。
- GitHub アクションから変数をエクスポートするには、「」を参照してください他のアクションで使用できるように GitHub 出力パラメータをエクスポートする。

前提条件

エクスポートする変数が定義されていることを確認します。詳細については、「変数の定義」を参照してください。

Visual

変数をエクスポートするには (ビジュアルエディタ)

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. ビジュアル を選択します。
7. ワークフロー図で、変数をエクスポートするアクションを選択します。
8. 出力 を選択します。
9. 変数 - オプション で、変数 を追加 を選択し、次の操作を行います。

   アクションでエクスポートする変数の名前を指定します。この変数は、同じアクションの `Inputs`または `Steps`セクションで既に定義されている必要があります。
10. (オプション) Validate を選択して、コミットする前にワークフローの YAML コードを検証します。
11. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

YAML

変数をエクスポートするには (YAML エディタ)

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。

5. [編集] を選択します。
6. YAML を選択します。
7. 変数をエクスポートするアクションで、次のようなコードを追加します。

```
action-name:
  Outputs:
    Variables:
      - Name: variable-name
```

その他の例については、「ユーザー定義変数の例」を参照してください。

8. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
9. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

変数を定義するアクションでの変数の参照

以下の手順に従って、変数を定義する アクションで変数を参照します。

> **Note**
>
> GitHub アクションによって生成された変数を参照するには、「」を参照してください GitHub 出力パラメータの参照。

前提条件

参照する変数が定義されていることを確認します。詳細については、「変数の定義」を参照してください。

Visual

使用できません。YAML を選択して YAML の手順を表示します。

YAML

変数を定義するアクションで変数を参照するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。

2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. YAML を選択します。
7. 参照する変数を定義する CodeCatalyst アクションで、次の bash 構文を使用して変数を追加します。

   ```
   $variable-name
   ```

   例:

   ```
   MyAction:
       Configuration:
         Steps:
           - Run: $variable-name
   ```

   その他の例については、「ユーザー定義変数の例」を参照してください。詳細については、「」の「 アクションのリファレンス情報」を参照してくださいワークフロー YAML 定義。
8. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
9. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

別のアクションによる変数出力の参照

他のアクションによって出力される変数を参照するには、次の手順を使用します。

> **Note**
>
> GitHub アクションからの変数出力を参照するには、「」を参照してください GitHub 出力パラメータの参照。

前提条件

参照する変数がエクスポートされていることを確認します。詳細については、「他のアクションで使用できるように変数をエクスポートする」を参照してください。

Visual

使用できません。YAML を選択して、YAML の手順を表示します。

YAML

別のアクションによる変数出力を参照するには (YAML エディタ)

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. YAML を選択します。
7. CodeCatalyst アクションで、次の構文を使用して変数への参照を追加します。

```
${action-group-name.action-name.variable-name}
```

置換:

- *action-group-name* 変数を出力するアクションを含むアクショングループの名前。

> **Note**
>
> アクショングループ*action-group-name*がない場合、または変数が同じアクショングループのアクションによって生成される場合は、 を省略できます。

- *action-name* 変数を出力するアクションの名前。
- *variable-name* と変数の名前。

例:

```
MySecondAction:
```

```
    Configuration:
      Steps:
        - Run: ${MyFirstAction.TIMESTAMP}
```

その他の例については、「ユーザー定義変数の例」を参照してください。詳細については、アクションのワークフロー YAML 定義「」を参照してください。

8. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
9. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

シークレットの参照

ワークフロー定義ファイルでシークレットを参照する手順については、「」を参照してくださいシークレットの使用。

例については、例: シークレットの参照を参照してください。

ユーザー定義変数の例

次の例は、ワークフロー定義ファイルで変数を定義して参照する方法を示しています。

例

- 例: Inputs プロパティを使用した変数の定義
- 例: Steps プロパティを使用した変数の定義
- 例: Outputs プロパティを使用した変数のエクスポート
- 例: 同じアクションで定義された変数を参照する
- 例: 別のアクションで定義された変数の参照
- 例: シークレットの参照

例: Inputs プロパティを使用した変数の定義

次の例は、 Inputsセクション VAR2で VAR1と の 2 つの変数を定義する方法を示しています。

```
Actions:
  Build:
    Identifier: aws/build@v1
    Inputs:
```

```
      Variables:
      - Name: VAR1
        Value: "My variable 1"
      - Name: VAR2
        Value: "My variable 2"
```

例: Steps プロパティを使用した変数の定義

次の例は、 StepsセクションでDATE変数を明示的に定義する方法を示しています。

```
Actions:
  Build:
    Identifier: aws/build@v1
    Configuration:
      Steps:
        - Run: DATE=$(date +%m-%d-%y)
```

例: Outputs プロパティを使用した変数のエクスポート

次の例は、 REPOSITORY-URIと の 2 つの変数を定義しTIMESTAMP、 Outputsセクションを使用してエクスポートする方法を示しています。

```
Actions:
  Build:
    Identifier: aws/build@v1
    Inputs:
      Variables:
        - Name: REPOSITORY-URI
          Value: 111122223333.dkr.ecr.us-east-2.amazonaws.com/codecatalyst-ecs-image-
repo
    Configuration:
      Steps:
        - Run: TIMESTAMP=$(date +%m-%d-%y-%H-%m-%s)
    Outputs:
      Variables:
        - REPOSITORY-URI
        - TIMESTAMP
```

例: 同じアクションで定義された変数を参照する

次の例は、 でVAR1変数を指定しMyBuildAction、 を使用して同じアクションで参照する方法を示しています$VAR1。

```
Actions:
  MyBuildAction:
    Identifier: aws/build@v1
    Inputs:
      Variables:
        - Name: VAR1
          Value: my-value
    Configuration:
      Steps:
        - Run: $VAR1
```

例: 別のアクションで定義された変数の参照

次の例は、 でTIMESTAMP変数を指定しBuildActionA、 Outputsプロパティを使用してエクスポートし、 BuildActionBを使用して で参照する方法を示しています${BuildActionA.TIMESTAMP}。

```
Actions:
  BuildActionA:
    Identifier: aws/build@v1
    Configuration:
      Steps:
        - Run: TIMESTAMP=$(date +%m-%d-%y-%H-%m-%s)
    Outputs:
      Variables:
        - TIMESTAMP
  BuildActionB:
    Identifier: aws/build@v1
    Configuration:
      Steps:
        - Run: docker build -t my-ecr-repo/image-repo:latest .
        - Run: docker tag my-ecr-repo/image-repo:${BuildActionA.TIMESTAMP}

        # Specifying just '$TIMESTAMP' here will not work
        # because TIMESTAMP is not a variable
        # in the BuildActionB action.
```

例: シークレットの参照

次の例は、my-passwordシークレットを参照する方法を示しています。my-password はシークレットのキーです。このシークレットのキーと対応するパスワードの値は、ワークフロー定義ファイ

ルで使用する前に、 CodeCatalyst コンソールのシークレットページで指定する必要があります。詳細については、「ワークフローでのシークレットの設定と使用」を参照してください。

```
Actions:
  BuildActionA:
    Identifier: aws/build@v1
    Configuration:
      Steps:
        - Run: curl -u LiJuan:${Secrets.my-password} https://example.com
```

事前定義された変数の使用

事前定義された変数は、ワークフローによって自動的に出力され、ワークフローアクションで使用できるようにするキーと値のペアです。

事前定義された変数は、任意のワークフローアクションで使用できます。

トピック

- 事前定義された変数の参照
- ワークフローが出力する事前定義された変数の決定
- 事前定義された変数の例

事前定義された変数の参照

事前定義された変数を参照するには、以下の手順に従います。

前提条件

など、参照する事前定義された変数の名前を決定しますCommitId。詳細については、「ワークフローが出力する事前定義された変数の決定」を参照してください。

Visual

使用できません。YAML を選択して YAML の手順を表示します。

YAML

事前定義された変数を参照するには (YAML エディタ)

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。

3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. YAML を選択します。
7. CodeCatalyst アクションで、次の構文を使用して事前定義された変数リファレンスを追加します。

```
${action-group-name.action-name-or-WorkflowSource.variable-name}
```

置換:

- *action-group-name* アクショングループの名前。

  > **Note**
  >
  > アクショングループ*action-group-name*がない場合、または変数が同じアクショングループ内のアクションによって生成される場合は、 を省略できます。

- *action-name-or-WorkflowSource* と :

  変数を出力するアクションの名前。

  または

  `WorkflowSource`変数が `BranchName`または `CommitId`変数の場合、。

- *variable-name* と変数の名前。

例:

```
MySecondAction:
    Configuration:
      Steps:
        - Run: echo ${MyFirstECSAction.cluster}
```

別の例を紹介します。

```
MySecondAction:
    Configuration:
      Steps:
        - Run: echo ${WorkflowSource.CommitId}
```

その他の例については、「事前定義された変数の例」を参照してください。詳細については、アクションのワークフロー YAML 定義「」を参照してください。

8. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
9. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

ワークフローが出力する事前定義された変数の決定

ワークフローが出力する事前定義された変数は、次の 2 つの方法で判断できます。

- ワークフローを 1 回実行します。実行が完了すると、ワークフローによって出力される変数が、実行の詳細ページの変数タブに表示されます。詳細については、「ワークフローの実行ステータスと詳細の表示」を参照してください。
- を参照してください事前定義された変数のリスト。このリファレンスには、事前定義された各変数の変数名 (キー) と値が一覧表示されます。

> **Note**
>
> ワークフローの変数の最大合計サイズは、 に記載されていますワークフローのクォータ。合計サイズが最大値を超えると、最大値に達した後に実行されるアクションが失敗する可能性があります。

事前定義された変数の例

次の例は、ワークフロー定義ファイルで事前定義された変数を参照する方法を示しています。

例

- 例：「」のCommitId事前定義された変数を参照する
- 例：「」のBranchName事前定義された変数を参照する

例：「」のCommitId事前定義された変数を参照する

次の例は、 MyBuildActionアクションでCommitId事前定義された変数を参照する方法を示しています。CommitId 変数は によって自動的に出力されます CodeCatalyst。

この例では、ビルドアクションで使用されている変数を示していますが、任意のアクションCommitIdで を使用できます。

```
MyBuildAction:
    Identifier: aws/build@v1
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      Steps:
      #Build Docker image and tag it with a commit ID
        - Run: docker build -t image-repo/my-docker-image:latest .
        - Run: docker tag image-repo/my-docker-image:${WorkflowSource.CommitId}
```

例：「」のBranchName事前定義された変数を参照する

次の例は、 CDKDeployアクションでBranchName事前定義された変数を参照する方法を示しています。BranchName 変数は によって自動的に出力されます CodeCatalyst。

この例では、AWS CDK デプロイアクションで使用されている変数を示していますが、任意のアクションBranchNameで を使用できます。

```
CDKDeploy:
    Identifier: aws/cdk-deploy@v1
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      StackName: app-stack-${WorkflowSource.BranchName}
```

事前定義された変数のリスト

以下のセクションを参照して、 CodeCatalyst アクションによって自動的に生成される事前定義された変数を表示します。

> **Note**
>
> このリストには、 CodeCatalyst ソース およびCodeCatalyst アクション によって出力される事前定義された変数のみが含まれます。アクションや CodeCatalyst ラボアクションなど、他のタイプの GitHub アクションを使用している場合は、代わりに「」を参照してくださいワークフローが出力する事前定義された変数の決定。

リスト

> **Note**
>
> すべての CodeCatalyst アクションが事前定義された変数を生成するわけではありません。アクションがリストにない場合、変数は生成されません。

- ソースによって生成される変数 (BranchName「」およびCommidId「」)
- AWS CloudFormation 「スタックのデプロイ」アクションによって生成される変数
- 「Amazon ECS へのデプロイ」アクションによって生成される変数
- 「Kubernetes クラスターにデプロイ」アクションによって生成される変数
- AWS CDK 「デプロイ」アクションによって生成される変数
- AWS CDK 「ブートストラップ」アクションによって生成される変数
- AWS Lambda 「呼び出し」アクションによって生成される変数
- 「Amazon ECS タスク定義のレンダリング」アクションによって生成される変数

ワークフローでのシークレットの設定と使用

ワークフローでは、認証情報などの機密データを使用する必要がある場合があります。シークレットを含むリポジトリへのアクセス権を持つすべてのユーザーがそれらの値を表示できるため、これらの値をリポジトリ内の任意の場所にプレーンテキストで保存することは避ける必要があります。同様に、これらの値はリポジトリ内のファイルとして表示されるため、ワークフロー定義で直接使用しないでください。を使用すると CodeCatalyst、プロジェクトにシークレットを追加し、ワークフロー定義ファイルでシークレットを参照することで、これらの値を保護できます。アクションごとに最大 5 つのシークレットを持つことができることに注意してください。

> **Note**
>
> シークレットは、ワークフロー定義ファイル内のパスワードと機密情報を置き換えるためにのみ使用できます。

トピック

シークレットの作成

シークレットを作成するには、次の手順に従います。シークレットには、非表示にする機密情報が含まれています。

> **Note**
>
> シークレットはアクションに表示され、ファイルに書き込まれる際にマスクされません。

シークレットを作成する

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. ナビゲーションペインで CI/CD を選択し、シークレット を選択します。
3. [シークレットの作成] を選択します。
4. 次の情報を入力します。

   名前

   シークレットの名前を入力します。

   値

   シークレットの値を入力します。これは、非表示にする機密情報です。デフォルトでは、値は表示されません。値を表示するには、値を表示 を選択します。

説明

(オプション) シークレットの説明を入力します。

5. [作成] を選択します。

シークレットの編集

シークレットを編集するには、次の手順に従います。

シークレットを編集するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. ナビゲーションペインで CI/CD を選択し、シークレット を選択します。
3. シークレットリストで、編集するシークレットを選択します。
4. [編集] を選択します。
5. 次のプロパティを編集します。

   値

   シークレットの値を入力します。これは、非表示にする値です。デフォルトでは、値は表示されません。

   説明

   (オプション) シークレットの説明を入力します。
6. [保存] を選択します。

シークレットの使用

ワークフローアクションでシークレットを使用するには、シークレットの参照識別子を取得し、その識別子をワークフローアクションで使用する必要があります。

トピック

- シークレットの識別子の取得
- ワークフローでシークレットを参照する

シークレットの識別子の取得

シークレットの参照識別子を取得するには、次の手順を使用します。この識別子をワークフローに追加します。

シークレットの参照識別子を取得するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. ナビゲーションペインで CI/CD を選択し、シークレット を選択します。
3. シークレットのリストで、使用するシークレットを見つけます。
4. リファレンス ID 列で、シークレットの識別子をコピーします。リファレンス ID の構文は次のとおりです。

```
${Secrets.<name>}
```

ワークフローでシークレットを参照する

ワークフロー内のシークレットを参照するには、次の手順に従います。

シークレットを参照するには

1. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
2. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
3. [編集] を選択します。
4. YAML を選択します。
5. シークレットの識別子を使用するように YAML を変更します。例えば、 `curl` コマンドでシークレットとして保存されているユーザー名とパスワードを使用するには、次のようなRunコマンドを使用します。

```
- Run: curl -u <username-secret-identifier>:<password-secret-identifier> https://
example.com
```

6. (オプション) Validate を選択して、コミットする前にワークフローの YAML コードを検証します。
7. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

シークレットの削除

シークレットとシークレット参照識別子を削除するには、次の手順に従います。

> **Note**
>
> シークレットを削除する前に、すべてのワークフローアクションからシークレットの参照識別子を削除することをお勧めします。参照識別子を削除せずにシークレットを削除すると、次回実行されるときにアクションは失敗します。

ワークフローからシークレットの参照識別子を削除するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
3. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
4. [編集] を選択します。
5. YAML を選択します。
6. ワークフローで次の文字列を検索します。

   ```
   ${Secrets.
   ```

   これにより、すべてのシークレットのすべての参照識別子が検索されます。
7. 選択したシークレットの参照識別子を削除するか、プレーンテキストの値に置き換えます。
8. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
9. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

シークレットを削除するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. ナビゲーションペインで CI/CD を選択し、シークレット を選択します。
3. シークレットリストで、削除するシークレットを選択します。

4. [削除] を選択します。
5. **delete** を入力して削除を確定します。
6. [削除] をクリックします。

ワークフローステータスの表示

ワークフローのステータスを表示して、対処する必要があるワークフロー設定の問題があるかどうかを確認したり、開始に失敗した実行のトラブルシューティングを行ったりできます。 CodeCatalyst は、ワークフローの基盤となるワークフロー定義ファイル を作成または更新するたびに、ワークフローのステータスを評価します。

> **Note**
>
> ワークフローの実行ステータスを表示することもできます。これは、ワークフローのステータスとは異なります。詳細については、「ワークフローの実行ステータスと詳細の表示」を参照してください。

考えられるワークフロー状態のリストについては、「」を参照してくださいワークフローの状態。

ワークフローのステータスを表示するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ステータスを表示するワークフローを検索します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。

   ステータスは、ワークフローとともにリストに表示されます。
5. (オプション) ワークフローの名前を選択し、ワークフロー定義フィールドを見つけます。ワークフローのステータスが表示されます。

ワークフローのクォータ

次の表に、Amazon のワークフローのクォータと制限を示します CodeCatalyst。

Amazon のクォータの詳細については、 CodeCatalyst「」を参照してくださいのクォータ CodeCatalyst。

スペースあたりのワークフローの最大数	800
ワークフロー定義ファイルの最大サイズ	256 KB
1 つのソースイベントで処理されるワークフローファイルの最大数	50
1 つのソースイベントで処理されるファイルの最大数	4,000
スペースあたりのアクティブなフリートの最大数	10
フリートあたりのアクティブなコンピューティングインスタンスの最大数	20
アクションあたりの入力アーティファクトの最大数	10
アクションあたりの出力アーティファクトの最大数	10
1 つのアクションの出力変数の最大合計サイズ	120 KB
出力変数値の最大長	値を出力するアクションに応じて、500 文字以上。 **Note** 値は、アクションの制限を超えた場合に切り捨てることができます。
ワークフローの実行中に生成されたアーティファクトを保持する最大日数	30
アクションあたりのレポートの最大数	50

テストレポートあたりのテストケースの最大数	20,000
コードカバレッジレポートあたりの最大ファイル数	20,000
レポートあたりのソフトウェアコンポジション分析結果の最大数	20,000
静的分析レポートあたりの最大ファイル数	20,000
スペースあたりの同時ワークフロー実行の最大数	100
ワークフローあたりのアクションの最大数	50
ワークフローごとに同時に実行されるアクションの最大数	50
スペースごとに同時に実行されるアクションの最大数	200
アクションを実行できる最大時間	ビルドアクションとテストアクションの場合、タイムアウトは 8 時間です。 他のすべてのアクションでは、タイムアウトは 1 時間です。
スペースあたりの AWS アカウント に関連付けられた環境の最大数	5,000
アクションあたりのシークレットの最大数	5
スペースあたりのシークレットの最大数	500,000

ワークフロー実行状態

ワークフロー実行は、次のいずれかの状態になります。

- 成功 – ワークフローの実行は正常に処理されました。

- Failed – ワークフロー実行の 1 つ以上のアクションが失敗しました。
- 進行中 — ワークフロー実行は現在処理中です。
- Stopped – 進行中のワークフローの実行を停止したユーザー。
- Stopping – ワークフローの実行は現在停止中です。
- キャンセル済み – 実行の進行中に関連付けられたワークフローが削除または更新された CodeCatalyst ため、ワークフロー実行が によってキャンセルされました。
- 置き換え済み – 置き換え済み実行モード を設定している場合にのみ発生します。ワークフロー実行は、後のワークフロー実行がそれに取って代わった CodeCatalyst ため、 によってキャンセルされました。

ワークフローの状態

ワークフローには、次のいずれかの状態があります。

- 有効 – ワークフローは実行可能で、トリガー によってアクティブ化できます。

  ワークフローを有効としてマークするには、次の両方の条件が満たされている必要があります。

  - ワークフロー定義ファイルは有効である必要があります。
  - ワークフローには、現在のブランチのファイルを使用して実行されるトリガー、プッシュトリガー、またはプッシュトリガーがあってはなりません。詳細については、「分岐時のトリガーに関する考慮事項」を参照してください。
- 無効 – ワークフローの定義ファイルが無効です。ワークフローは手動で実行することも、トリガーを使用して自動的に実行することもできません。無効なワークフローは、ワークフロー定義とともにコンソールにエラーメッセージ (または同様のメッセージ) が ***n*** つ表示されます。CodeCatalyst

  ワークフローを無効としてマークするには、次の条件が満たされている必要があります。

  - ワークフロー定義ファイルは誤って設定されている必要があります。

    誤って設定されたワークフロー定義ファイルを修正するには、「」を参照してください「ワークフロー定義に *n* エラーがある」エラーを修正するにはどうすればよいですか？。
- 非アクティブ – ワークフロー定義は有効ですが、手動で実行することも、トリガーを使用して自動的に実行することもできません。

  ワークフローを非アクティブとしてマークするには、次の両方の条件が満たされている必要があります。

- ワークフロー定義ファイルは有効である必要があります。
- ワークフロー定義ファイルには、ワークフロー定義ファイルが存在するブランチとは異なるブランチを指定するプッシュトリガーが含まれている必要があります。詳細については、「分岐時のトリガーに関する考慮事項」を参照してください。

  ワークフローを非アクティブからアクティブに切り替えるには、「」を参照してください「ワークフローは非アクティブです」というメッセージを修正するにはどうすればよいですか？。

> **Note**
>
> ワークフローで後で削除するリソース (パッケージリポジトリなど) CodeCatalyst が指定されている場合、この変更は検出されず、ワークフローを引き続き有効としてマークします。これらのタイプの問題は、ワークフローの実行時に検出されます。

ワークフロー YAML 定義

ワークフロー定義ファイルのリファレンスドキュメントを次に示します。

ワークフロー定義ファイルは、ワークフローを記述する YAML ファイルです。ファイルは、ソースリポジトリ のルートにある~/.codecatalyst/workflows/フォルダに保存されます。ファイルには、.yml または .yaml 拡張子を付けることができます。

ワークフロー定義ファイルを作成および編集するには、vim などのエディタを使用するか、CodeCatalyst コンソールのビジュアルエディタまたは YAML エディタを使用できます。詳細については、「 CodeCatalyst コンソールのビジュアルエディタと YAML エディタの使用」を参照してください。

> **Note**
>
> 後続の YAML プロパティのほとんどには、対応する UI 要素がビジュアルエディタにあります。UI 要素を検索するには、Ctrl+F を使用します。 要素は、関連付けられた YAML プロパティとともに一覧表示されます。

トピック

- ワークフロー定義ファイルの例

- 構文のガイドラインと規則
- 最上位のプロパティ

ワークフロー定義ファイルの例

以下は、シンプルなワークフロー定義ファイルの例です。これには、いくつかの最上位プロパティ、Triggersセクション、および Buildと の 2 つのアクションを含む Actionsセクションが含まれますTest。詳細については、「ワークフロー定義ファイルについて」を参照してください。

```
Name: MyWorkflow
SchemaVersion: 1.0
RunMode: QUEUED
Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  Build:
    Identifier: aws/build@v1
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      Steps:
        - Run: docker build -t MyApp:latest .
  Test:
    Identifier: aws/managed-test@v1
    DependsOn:
      - Build
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      Steps:
        - Run: npm install
        - Run: npm run test
```

構文のガイドラインと規則

このセクションでは、ワークフロー定義ファイルの構文ルールと、このリファレンスドキュメントで使用される命名規則について説明します。

YAML 構文ガイドライン

ワークフロー定義ファイルは YAML で記述され、YAML 1.1 仕様 に従っているため、その仕様で許可されているものはワークフロー YAML でも許可されます。YAML を初めて使用する場合は、有効な YAML コードを提供していることを確認するための簡単なガイドラインをいくつか紹介します。

- 大文字と小文字の区別: ワークフロー定義ファイルは大文字と小文字が区別されるため、このドキュメントに示されている大文字と小文字を使用してください。
- 特殊文字 : プロパティ値には、、{、、}、、、[]、、、、、?|-< * 、 > #、、=、!、%@、:、 ` のいずれかの特殊文字を含む引用符または二重引用符を使用することをお勧めします。 ,

  引用符を含めない場合、前述の特殊文字が予期しない解釈される可能性があります。
- プロパティ名 : プロパティ名 (プロパティ値 ではなく) は、英数字 (a ~ z、A ~ Z、0 ~ 9)、ハイフン (-)、アンダースコア (_) に制限されます。スペースは使用できません。プロパティ名で特殊文字やスペースを有効にするために、引用符や二重引用符を使用することはできません。

  許可されていません :

  ```
  'My#Build@action'

  My#Build@action

  My Build Action
  ```

  許可 :

  ```
  My-Build-Action_1
  ```
- エスケープコード : プロパティ値にエスケープコード (\nや など\t) が含まれている場合は、次のガイドラインに従ってください。
  - 一重引用符を使用してエスケープコードを文字列として返します。例えば、 は文字列 'my string \n my string'を返しますmy string \n my string。
  - 二重引用符を使用してエスケープコードを解析します。例えば、 は"my string \n my new line"以下を返します。

    ```
    my string
    my new line
    ```
- コメント : コメントの前に を付けます#。

例：

```
Name: MyWorkflow
# This is a comment.
SchemaVersion: 1.0
```

- トリプルダッシュ (`---`) : YAML code. CodeCatalyst ignores `---`では、 の後にすべてを無視します`---`。

命名規則

このガイドでは、 プロパティと セクションという用語を使用して、ワークフロー定義ファイルの主な項目を参照します。

- プロパティは、コロン (`:`) を含むすべての項目です。例えば、次のコードスニペットでは、、`Name`、、`SchemaVersion`、`RunMode`、および のプロパティはすべて `Triggers` TypeですBranches。
- セクションは、サブプロパティを持つ任意のプロパティです。次のコードスニペットには、1 つのTriggersセクションがあります。

> **Note**
>
> このガイドでは、「セクション」は「プロパティ」と呼ばれることがあり、コンテキストに応じてその逆を可視化します。

```
Name: MyWorkflow
SchemaVersion: 1.0
RunMode: QUEUED
Triggers:
  - Type: PUSH
    Branches:
      - main
```

最上位のプロパティ

以下は、ワークフロー定義ファイルの最上位プロパティのリファレンスドキュメントです。

```
# Name
Name: workflow-name

# Schema version
SchemaVersion: 1.0

# Run mode
RunMode: QUEUED|SUPERSEDED|PARALLEL

# Compute
Compute:
...

# Triggers
Triggers:
...

# Actions
Actions:
...
```

Name

(必須)

ワークフローの名前。ワークフロー名はワークフローリストに表示され、通知とログに記載されています。ワークフロー名とワークフロー定義ファイル名は一致することも、異なる名前を付けることもできます。ワークフロー名は一意である必要はありません。ワークフロー名は、英数字 (a～z、A～Z、0～9)、ハイフン (-)、アンダースコア (_) に制限されています。スペースは使用できません。引用符を使用してワークフロー名の特殊文字やスペースを有効にすることはできません。

対応する UI: ビジュアルエディタ/ワークフロープロパティ/ワークフロー名

SchemaVersion

(必須)

ワークフロー定義のスキーマバージョン。現在、有効な値は `1.0` のみです。

対応する UI: なし

RunMode

(オプション)

が複数の実行 CodeCatalyst を処理する方法。次のいずれかの値を使用できます。

- QUEUED – 複数の実行がキューに入れられ、順番に実行されます。キューには最大 50 回実行できます。
- SUPERSEDED – 複数の実行がキューに入れられ、順番に実行されます。キューは 1 つの実行しかできないため、2 つの実行が同じキューにまとめられた場合、後の実行は以前の実行よりも優先され (引き継がれます) 、以前の実行はキャンセルされます。
- PARALLEL – 複数の実行が同時に実行されます。

このプロパティを省略した場合、デフォルトは ですQUEUED。

詳細については、「実行のキューイング動作の設定」を参照してください。

対応する UI: ビジュアルエディタ/ワークフロープロパティ/アドバンスト/実行モード

Compute

(オプション)

ワークフローアクションを実行するために使用されるコンピューティングエンジン。コンピューティングはワークフローレベルまたはアクションレベルで指定できますが、両方を指定することはできません。ワークフローレベルで指定すると、コンピューティング設定はワークフローで定義されたすべてのアクションに適用されます。ワークフローレベルでは、同じインスタンスで複数のアクションを実行することもできます。詳細については、「アクション間でのコンピューティングの共有」を参照してください。

コンピューティングの詳細については、「」を参照してくださいワークフローのコンピューティング環境とランタイム環境の Docker イメージの設定。

対応する UI: なし

```
Name: MyWorkflow
SchemaVersion: 1.0
...
Compute:
```

```
  Type: EC2 | Lambda
  Fleet: fleet-name
  SharedInstance: true | false
```

Type

(Compute/Type)

(Computeが設定されている場合は必須です)

コンピューティングエンジンのタイプ。次のいずれかの値を使用できます。

- EC2 (ビジュアルエディタ) または EC2 (YAML エディタ)

  アクション実行中の柔軟性のために最適化されました。
- Lambda (ビジュアルエディタ) または Lambda (YAML エディタ)

  アクションの起動速度を最適化しました。

コンピューティングタイプの詳細については、「コンピューティングタイプ」を参照してください。

対応する UI: ビジュアルエディタ/ワークフロープロパティ/アドバンスト/コンピューティングタイプ

Fleet

(Compute/Fleet)

(オプション)

ワークフローまたはワークフローアクションを実行するマシンまたはフリートを指定します。オンデマンドフリートでは、アクションが開始されると、ワークフローは必要なリソースをプロビジョニングし、アクションが終了するとマシンは破棄されます。オンデマンドフリートの例: Linux.x86-64.Large、Linux.x86-64.XLarge。オンデマンドフリートの詳細については、「」を参照してくださいオンデマンドフリートのプロパティ。

プロビジョニングされたフリートでは、ワークフローアクションを実行するように一連の専用マシンを設定します。これらのマシンはアイドル状態のままで、すぐにアクションを処理できます。プロビジョニングされたフリートの詳細については、「」を参照してくださいプロビジョニングされたフリートのプロパティ。

Fleet を省略した場合、デフォルトは ですLinux.x86-64.Large。

コンピューティングフリートの詳細については、「」を参照してくださいコンピューティングフリート。

対応する UI: ビジュアルエディタ/ワークフロープロパティ/アドバンスト/コンピューティングフリート

SharedInstance

(Compute/SharedInstance)

(オプション)

アクションのコンピューティング共有機能を指定します。コンピューティング共有では、ワークフロー内のアクションは同じインスタンス (ランタイム環境イメージ) で実行されます。次のいずれかの値を使用できます。

- TRUE は、ランタイム環境イメージがワークフローアクション間で共有されることを意味します。
- FALSE は、ワークフロー内のアクションごとに個別のランタイム環境イメージが開始されて使用されるため、追加の設定なしでアーティファクトや変数などのリソースを共有することはできません。

コンピューティング共有の詳細については、「」を参照してくださいアクション間でのコンピューティングの共有。

対応する UI: なし

Triggers

(オプション)

このワークフローの 1 つ以上のトリガーのシーケンス。トリガーが指定されていない場合は、ワークフローを手動で開始する必要があります。

トリガーについての詳細は、「トリガーを使用したワークフローの自動実行の開始」を参照してください。

対応する UI: ビジュアルエディタ/ワークフロー図/トリガー

```
Name: MyWorkflow
SchemaVersion: 1.0
...
```

```
Triggers:
  - Type: PUSH
    Branches:
      - branch-name
    FilesChanged:
      - folder1/file
      - folder2/

  - Type: PULLREQUEST
    Events:
      - OPEN
      - CLOSED
      - REVISION
    Branches:
      - branch-name
    FilesChanged:
      - file1.txt

  - Type: SCHEDULE
    # Run the workflow at 10:15 am (UTC+0) every Saturday
    Expression: "15 10 ? * 7 *"
    Branches:
      - branch-name
```

Type

(Triggers/Type)

(が設定されている場合Triggersは必須です)

トリガーのタイプを指定します。次のいずれかの値を使用できます。

- プッシュ (ビジュアルエディタ) または PUSH (YAML エディタ)

  プッシュトリガーは、変更がソースリポジトリにプッシュされたときにワークフロー実行を開始します。ワークフローの実行では、プッシュ先のブランチ (つまり、送信先ブランチ) 内のファイルを使用します。
- プルリクエスト (ビジュアルエディタ) または PULLREQUEST (YAML エディタ)

  プルリクエストトリガーは、ソースリポジトリでプルリクエストがオープン、更新、またはクローズされたときにワークフロー実行を開始します。ワークフローの実行では、プル元のブランチ (ソースブランチ) 内のファイルを使用します。

- スケジュール (ビジュアルエディタ) または SCHEDULE (YAML エディタ)

  スケジュールトリガーは、指定した cron 式で定義されたスケジュールでワークフロー実行を開始します。ブランチの ファイルを使用して、ソースリポジトリ内のブランチごとに個別のワークフロー実行が開始されます。(トリガーがアクティブ化されるブランチを制限するには、ブランチフィールド (ビジュアルエディタ) またはBranchesプロパティ (YAML エディタ) を使用します。)

  スケジュールトリガーを設定するときは、次のガイドラインに従ってください。

  - ワークフローごとに 1 つのスケジュールトリガーのみを使用します。
  - CodeCatalyst スペースに複数のワークフローを定義している場合は、同時に開始するように 10 個までスケジュールすることをお勧めします。
  - トリガーの cron 式は、実行間隔を十分に設定してください。詳細については、「Expression」を参照してください。

例については、「トリガーの例」を参照してください。

対応する UI: ビジュアルエディタ/ワークフロー図/トリガー/トリガータイプ

Events

(Triggers/Events)

(トリガーが Typeに設定されている場合は必須ですPULLREQUEST)

ワークフロー実行を開始するプルリクエストイベントのタイプを指定します。有効な値は次のとおりです。

- プルリクエストが作成される (ビジュアルエディタ) または OPEN (YAML エディタ)

  ワークフローの実行は、プルリクエストの作成時に開始されます。

- プルリクエストがクローズされている (ビジュアルエディタ) または CLOSED (YAML エディタ)

  ワークフロー実行は、プルリクエストがクローズされたときに開始されます。CLOSED イベントの動作はトリッキーで、例から最もよく理解できます。詳細については、「例: プル、ブランチ、および「CLOSED」イベントを含むトリガー」を参照してください。

- プルリクエスト (ビジュアルエディタ) または (YAML エディタ) に新しいリビジョンが加えられました REVISION

ワークフロー実行は、プルリクエストのリビジョンが作成されると開始されます。最初のリビジョンは、プルリクエストの作成時に作成されます。その後、プルリクエストで指定されたソースブランチに誰かが新しいコミットをプッシュするたびに、新しいリビジョンが作成されます。プルリクエストトリガーREVISIONにイベントを含めると、 が のスーパーセットであるため、OPENイベントREVISIONを省略できますOPEN。

同じプルリクエストトリガーで複数のイベントを指定できます。

例については、「トリガーの例」を参照してください。

対応する UI: プルリクエストのビジュアルエディタ/ワークフロー図/トリガー/イベント

Branches

(Triggers/Branches)

(オプション)

ワークフロー実行を開始するタイミングを知るために、トリガーがモニタリングするソースリポジトリ内のブランチを指定します。正規表現パターンを使用してブランチ名を定義できます。例えば、main.*を使用して、 で始まるすべてのブランチを照合しますmain。

指定するブランチは、トリガータイプによって異なります。

- プッシュトリガーでは、 にプッシュするブランチ、つまり送信先ブランチを指定します。一致するブランチ内のファイルを使用して、一致するブランチごとに 1 つのワークフロー実行が開始されます。

  例: main.*、mainline
- プルリクエストトリガーでは、 にプッシュするブランチ、つまり送信先ブランチを指定します。ワークフロー定義ファイルとソースブランチ内のソースファイル (一致したブランチではない) を使用して、一致したブランチごとに 1 回のワークフロー実行が開始されます。

  例: main.*、mainline、 v1\-.* (で始まるブランチに一致v1-)
- スケジュールトリガーには、スケジュールされた実行で使用するファイルを含むブランチを指定します。一致するブランチごとに 1 つのワークフロー実行が開始され、一致するブランチのワークフロー定義ファイルとソースファイルが使用されます。

  例: main.*、version\-1\.0

> **Note**
>
> ブランチを指定しない場合、トリガーはソースリポジトリ内のすべてのブランチをモニタリングし、ワークフロー定義ファイルとソースファイルを使用してワークフロー実行を開始します。
>
> - プッシュ先のブランチ (プッシュトリガーの場合) 。詳細については、「例: シンプルなコードプッシュトリガー」を参照してください。
> - プル元のブランチ (プルリクエストトリガーの場合) 。詳細については、「例: シンプルなプルリクエストトリガー」を参照してください。
> - すべてのブランチ (スケジュールトリガー用) 。ソースリポジトリのブランチごとに 1 回のワークフロー実行が開始されます。詳細については、「例: シンプルなスケジュールトリガー」を参照してください。

ブランチとトリガーの詳細については、「」を参照してください分岐時のトリガーに関する考慮事項。

その他の例については、「トリガーの例」を参照してください。

対応する UI: ビジュアルエディタ/ワークフロー図/トリガー/ブランチ

FilesChanged

(Triggers/FilesChanged)

(トリガーが PUSH、、または に設定されている場合TypeはオプションPULLREQUESTです。 トリガーが に設定されている場合Type、サポートされていませんSCHEDULE。)

ワークフロー実行を開始するタイミングを知るために、トリガーがモニタリングするソースリポジトリ内のファイルまたはフォルダを指定します。正規表現を使用して、ファイル名またはパスを一致させることができます。

例については、「トリガーの例」を参照してください。

対応する UI: ビジュアルエディタ/ワークフロー図/トリガー/ファイルの変更

Expression

(Triggers/Expression)

(トリガーが Typeに設定されている場合は必須ですSCHEDULE)

スケジュールされたワークフロー実行をいつ実行するかを記述する cron 式を指定します。

の Cron 式では、次の 6 つのフィールド構文 CodeCatalyst を使用します。各フィールドはスペースで区切られます。

 # 時間 days-of-month # days-of-week#

cron 式の例

分	時間	曜日	月	曜日	年	意味
0	0	?	*	MON-FRI	*	毎週月曜日から金曜日の午前 0 時 (UTC+0) にワークフローを実行します。
0	2	*	*	?	*	ワークフローを毎日午前 2 時 (UTC+0) に実行します。
15	22	*	*	?	*	ワークフローを毎日午後 10:15 (UTC+0) に実行します。
0/30	22-2	?	*	土 - 日	*	ワークフローは、土曜日から日曜日の開

分	時間	曜日	月	曜日	年	意味
						始日の午後 10 時から翌日の午前 2 時 (UTC+0) の間、30 分ごとに実行されます。
45	13	L	*	?	2023-2027	2023 年から 2027 年までの月の最終日の午後 1 時 45 分 (UTC+0) にワークフローを実行します。

で cron 式を指定するときは CodeCatalyst、次のガイドラインに従ってください。

- SCHEDULE トリガーごとに 1 つの cron 式を指定します。
- YAML エディタで cron 式を二重引用符 (") で囲みます。
- 協定世界時 (UTC) で時刻を指定します。その他のタイムゾーンはサポートされていません。
- 実行の合間に 30 分以上を設定します。より高速なケイデンスはサポートされていません。
- *days-of-month* または *days-of-week*フィールドを指定しますが、両方を指定することはできません。いずれかのフィールドに値またはアスタリスク (*) を指定する場合は、もう一方のフィールドで疑問符 (?) を使用する必要があります。アスタリスクは「all」、疑問符は「any」を意味します。

cron 式のその他の例と、?、、 *などのワイルドカードに関する情報についてはL、Amazon EventBridge ユーザーガイドの「Cron 式のリファレンス」を参照してください。 EventBridge と の Cron 式はまったく同じように CodeCatalyst 動作します。

スケジュールトリガーの例については、「」を参照してくださいトリガーの例。

対応する UI: ビジュアルエディタ/ワークフロー図/トリガー/スケジュール

アクション

このワークフローの一連の 1 つ以上のアクション。 は、ビルドアクションやテストアクションなど、さまざまなタイプの機能を提供する複数のアクションタイプ CodeCatalyst をサポートしています。各アクションタイプには次のものがあります。

- アクションの一意のハードコードされた ID を示す Identifierプロパティ。例えば、 はビルドアクションaws/build@v1を識別します。
- アクションに固有のプロパティを含むConfigurationセクション。

各アクションタイプの詳細については、「」を参照してくださいアクションタイプ。アクションタイプ トピックには、各アクションのドキュメントへのリンクがあります。

以下は、ワークフロー定義ファイル内のアクションとアクショングループの YAML リファレンスです。

```
Name: MyWorkflow
SchemaVersion: 1.0
...
Actions:
  action-or-gate-name:
    Identifier: identifier
    Configuration:
    ...
  #Action groups
  action-group-name:
    Actions:
      ...
```

action-or-gate-name

(Actions/*action-or-gate-name*)

(必須)

action-name を、アクションに付ける名前に置き換えます。アクション名はワークフロー内で一意である必要があり、英数字、ハイフン、アンダースコアのみを含める必要があります。構文ルールの詳細については、「」を参照してくださいYAML 構文ガイドライン。

制限を含むアクションの命名方法の詳細については、「」を参照してくださいaction-or-gate-name。

対応する UI: ビジュアルエディタ/*######*/設定タブ/アクション名またはアクション表示名

action-group-name

(Actions/*action-group-name*)

(オプション)

アクショングループには、1 つ以上のアクションが含まれます。アクションをアクショングループにグループ化すると、ワークフローを整理しやすくなり、異なるグループ間の依存関係を設定することもできます。

をアクショングループに付ける名前*action-group-name*に置き換えます。アクショングループ名はワークフロー内で一意である必要があり、英数字、ハイフン、アンダースコアのみを含める必要があります。構文ルールの詳細については、「」を参照してくださいYAML 構文ガイドライン。

アクショングループの詳細については、「」を参照してくださいアクションをアクショングループにグループ化する。

対応する UI: なし

で問題のある作業を追跡して整理する CodeCatalyst

では CodeCatalyst、機能、バグ、およびプロジェクトに関連するその他の作業をモニタリングできます。各作業は、問題 と呼ばれる個別のレコードに保持されます。タスクのチェックリストを追加することで、問題を小さな目標に分割できます。各問題には、説明、担当者、ステータス、およびその他のプロパティがあり、検索、グループ化、フィルタリングできます。デフォルトのビューを使用して問題を表示することも、カスタムフィルタリング、ソート、グループ化を使用して独自のビューを作成することもできます。問題に関連する概念の詳細については、「」を参照してください問題の概念。最初の問題を作成する方法については、「」を参照してください での問題の作成 CodeCatalyst。

問題を使用するチームで考えられるワークフローの 1 つを次に示します。

Jorge Souza は、プロジェクトで作業している開発者です。同僚のプロジェクトメンバーである Li Juan、Mateo Jackson、Wang Xiulan が協力して、どのような作業を行う必要があるかを判断します。毎日、Wang Xiulan が率いる同期会議を開催しています。彼らは、ボードのチームビューの 1 つに移動してボードを立ち上げます。ビューを作成することで、ユーザーとチームはフィルター、グループ化、問題のソートを保存して、指定した基準を満たす問題を簡単に表示できます。そのビューには、各デベロッパーの最も重要な問題と問題のステータスを示すために、担当者別にグループ化され、優先度別にソートされた問題が含まれています。Jorge には完了するタスクが割り当てられるため、タスクごとに問題を作成して作業を計画します。問題を作成するときに、Jorge は適切なステータス 、優先度 、および作業見積もりの労力を選択できます。より大きな問題の場合、Jorge は作業をより小さな目標に分割するために、問題にタスクを追加します。Jorge はすぐに開始する予定がないため、バックログ などのドラフトステータスで問題を作成します。ドラフトステータスの問題は、計画および優先順位付けを行うドラフトビューに表示されます。Jorge は、作業を開始する準備ができたら、ステータスを別のカテゴリ (未開始 、開始 、完了) のステータスに更新して、対応する問題をボードに移動します。各タスクが処理されている間、チームはタイトル、ステータス、担当者、ラベル、優先度、推定でフィルタリングして、指定されたパラメータに一致する特定の問題または類似の問題を見つけることができます。ボードを使用すると、Jorge と彼のチームは、各問題に対して完了したタスクの数を確認し、タスクが完了するまで各問題を 1 つのステータスから次のステータスにドラッグして day-to-day 進行状況を追跡できます。プロジェクトが進むにつれて、完了した問題は完了ステータスに蓄積されます。Wang Xiulan は、クイックアーカイブボタンを使用してアーカイブすることで、それらをビューから削除することを決定し、デベロッパーが現在および今後の作業に関連する問題に集中できるようにします。

作業を計画する際、プロジェクトに取り組んでいるデベロッパーは、ソート基準とグループ化基準を選択して、バックログからボードに移動する問題を見つけます。最も優先度の高い顧客リクエストに

基づいて問題をボードに追加する選択をする場合があるため、ボードを顧客リクエストラベルでグループ化し、優先度 でソートします。また、見積りでソートして、達成可能な作業量を確保することもできます。プロジェクトマネージャーの Saanvi Sarkar は、定期的にバックログを確認して整理し、プロジェクトの成功における各問題の重要性が優先度に正確に反映されるようにします。

トピック

問題の概念

問題の作成は、プロジェクト内で行われている作業を追跡するための迅速かつ効率的な方法です。問題を使用して、毎日の同期会議での作業の議論、作業の優先順位付けなどを行うことができます。

このページには、 の問題を効果的に使用するのに役立つ概念のリストが含まれています CodeCatalyst。

アクティブな問題

アクティブな問題とは、ドラフトステータスまたはアーカイブされていない問題です。つまり、アクティブな問題は、開始されていない、開始された、完了された のいずれかのステータスカテゴリにおけるステータスの問題です。ステータスとステータスカテゴリの詳細については、「」を参照してくださいステータスとステータスカテゴリ。

プロジェクト内のすべてのアクティブな問題は、デフォルトのアクティブな問題ビューから表示できます。

アーカイブされた問題

アーカイブされた問題は、プロジェクトに関連しなくなった問題です。例えば、問題が完了し、Done 列に表示する必要がなくなった場合や、エラーで作成された場合は、問題をアーカイブできます。アーカイブされた問題は、必要に応じてアーカイブ解除できます。

担当者

担当者とは、問題が割り当てられている人です。検索時にリストに表示されない場合は、プロジェクトに追加されていません。これらを追加するには、「」を参照してくださいプロジェクトへのユーザーの招待。問題に対して複数の担当者を有効にするには、「」を参照してください複数の担当者の有効化または無効化。複数の担当者に関する問題は、それぞれ担当者の 1 人を表す異なる色のアバターでボードに表示されます。

カスタム フィールド

カスタムフィールドを使用すると、プロジェクト内の問題を追跡および維持するニーズに応じて、問題のさまざまな属性をカスタマイズできます。例えば、ロードマップ作成用のフィールド、特定の期日、またはリクエスタフィールドを追加できます。

見積り

アジャイル開発では、見積もりはストーリーポイントと呼ばれます。問題のあいまいさと複雑さに加えて、問題の見積もりを使用して、必要な作業量を表すことができます。リスク、難易度、不明度が高い問題については、より高い見積りを使用することを検討してください。

推定タイプとその設定方法の詳細については、「」を参照してください問題労力の見積もりの設定。

問題

問題は、プロジェクトに関連する作業を追跡するレコードです。機能、タスク、バグ、またはプロジェクトに関連するその他の作業について問題を作成できます。アジャイル開発を使用している場合、問題はエピックやユーザーストーリーを記述することもできます。

ラベル

ラベルは、問題をグループ化、ソート、フィルタリングするために使用されます。新しいラベル名を入力するか、入力されたリストからいずれかのラベルを選択できます。このリストは、プロジェクトで最近使用されたラベルで構成されます。問題には複数のラベルがあり、1 つのラベルを問題から削除できます。ラベルをカスタマイズするには、「」を参照してくださいラベルを使用した作業の分類。

優先度

Priority とは、問題の重要度を指します。低 、中 、高 、優先度なし の 4 つのオプションがあります。

ステータスとステータスカテゴリ

ステータスは問題の現在の状態であり、開始から完了まで、ライフサイクルを通じて問題の進行状況をすばやくチェックするために使用されます。すべての問題には ステータスが必要で、各ステータスはステータスカテゴリ に属します。ステータスカテゴリは、ステータスの整理とデフォルトの問題ビューの入力に役立ちます。

には、5 つのデフォルトステータスと 4 つのステータスカテゴリがあります CodeCatalyst。他のステータスは作成できますが、他のステータスカテゴリは作成できません。次のリストには、デフォルトステータスとそのステータスカテゴリが括弧内に含まれています: バックログ (ドラフト)、To do (未開始)、In progress (開始)、In review (開始)、Done (完了)。

ステータスの操作の詳細については、「」を参照してくださいカスタムステータスの追跡作業。

タスク

タスクを問題に追加して、その問題の作業をさらに分類して整理できます。作成時に問題にタスクを追加したり、既存の問題にタスクを追加したりできます。問題を表示するときは、タスクの順序を変更、削除、または完了としてマークできます。

ビュー

CodeCatalyst プロジェクトの問題はビュー に表示されます。ビューは、リスト形式で問題を表示するグリッドビューでも、問題ステータス別に整理された列で問題がタイルとして表示されるボードビューでもかまいません。4 つのデフォルトビューがあり、カスタムグループ化、フィルタリング、ソートを使用して独自のビューを作成できます。次のリストには、4 つのデフォルトビューの詳細が含まれています。

- ドラフトビューは、現在処理されていない問題を示すグリッドビューです。ドラフトステータスカテゴリのステータスで作成された問題は、このビューに表示されます。このビューは、チームがどの問題がまだ定義されているか、割り当てられて作業されるのを待っているかを確認するために使用できます。
- アクティブな問題ビューは、現在処理中のすべての問題のボードビューです。Not started 、Started 、または Completed ステータスカテゴリのステータスに関する問題は、このビューに表示されます。
- 「すべての問題」ビューは、ドラフトとアクティブな問題の両方を含む、プロジェクト内のすべての問題を表示するグリッドビューです。
- アーカイブされたビューには、アーカイブされたすべての問題が表示されます。

問題のある作業の追跡

問題を使用して、プロジェクトでの作業を計画および追跡できます。各問題は、個別のレコードに保持されている作業です。問題はタスクに分割され、その問題の作業をさらに整理して追跡できます。また、問題間のリンクを作成して、関連する作業を追跡したり、作業の整理と分類に役立つラベルを追加したり、問題をグループ化したり、作業の優先順位を割り当てたり、作業がブロックされているかどうかを示すこともできます。

問題や一連の問題に取り組む準備ができたら、作業を見積もってユーザーに割り当てることができ、他のユーザーが作業とその進行状況を理解するのに役立つコメントを追加できます。また、問題をエクスポートして、含まれている情報を他の形式に取り込むこともできます。

での問題の作成 CodeCatalyst

開発チームは、作業の追跡と管理に役立つ問題を作成します。必要に応じて、プロジェクト内で問題を作成できます。例えば、コード内の変数の更新を追跡する問題を作成できます。プロジェクト内の他のユーザーに問題の割り当て、作業の追跡に役立つラベルの使用などを行うことができます。

以下の手順に従って、 で問題を作成します CodeCatalyst。

問題を作成するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 問題を作成するプロジェクトに移動します。
3. プロジェクトのホームページで、「問題の作成」を選択します。または、ナビゲーションペインで問題 を選択します。
4. 問題の作成 を選択します。

> **Note**
>
> グリッドビューを使用するときに、問題をインラインで追加することもできます。

5. 問題のタイトルを入力します。
6. (オプション) 説明 を入力します。Markdown を使用して書式を追加できます。
7. (オプション) 問題のステータス 、優先度 、および推定 を選択します。

> **Note**
>
> プロジェクトの推定設定が見積りを非表示 に設定されている場合、見積りフィールドはありません。

8. (オプション) 問題にタスクを追加します。タスクは、問題の作業をより小さな目標に分割するために使用できます。タスクを追加するには、「」 + 「タスクを追加」を選択します。次に、テキストフィールドにタスク名を入力し、Enter キーを押します。タスクを追加したら、チェックボックスを選択して完了としてマークするか、チェックボックスの左側からタスクを選択してドラッグして順序を変更できます。
9. (オプション) 既存のラベルを追加するか、新しいラベルを作成して + ラベルを追加 を選択して追加します。
   a. 既存のラベルを追加するには、リストからラベルを選択します。フィールドに検索語を入力すると、その語句を含むすべてのラベルをプロジェクトで検索できます。
   b. 新しいラベルを作成して追加するには、検索フィールドに作成するラベルの名前を入力し、Enter キーを押します。
10. (オプション) + 担当者を追加 を選択して、担当者を追加します。+ Add me を選択すると、自分自身を担当者としてすばやく追加できます。

> **Tip**
>
> Amazon Q に問題を割り当てることで、Amazon Q で問題の解決を試みることができます。詳細については、「チュートリアル: CodeCatalyst 生成 AI 機能を使用して開発作業を高速化する」を参照してください。この機能は、米国西部 (オレゴン) リージョンでのみ使用できます。
> この機能を使用するには、空間に対して生成 AI 機能を有効にする必要があります。詳細については、「生成 AI 機能の管理」を参照してください。

11. (オプション) 既存のカスタムフィールドを追加するか、新しいカスタムフィールドを作成します。問題には、複数のカスタムフィールドを含めることができます。
   a. 既存のカスタムフィールドを追加するには、リストからカスタムフィールドを選択します。フィールドに検索語を入力すると、プロジェクト内のその語を含むすべてのカスタムフィールドを検索できます。

b. 新しいカスタムフィールドを作成して追加するには、作成するカスタムフィールドの名前を検索フィールドに入力します。次に、作成するカスタムフィールドのタイプを選択し、値を設定します。

12. 問題の作成 を選択します。右下隅に通知が表示されます。問題が正常に作成された場合は、問題が正常に作成されたことを示す確認メッセージが表示されます。問題が正常に作成されなかった場合は、失敗の理由を示すエラーメッセージが表示されます。その後、再試行を選択して問題を編集して作成を再試行するか、破棄を選択して問題を破棄できます。どちらのオプションも通知を却下します。

> **Note**
>
> プルリクエストを作成するときに、プルリクエストを問題にリンクすることはできません。ただし、作成後に編集してプルリクエストへのリンクを追加できます。

Amazon Q に割り当てられた問題を作成および操作する際のベストプラクティス

問題を作成すると、一部の問題が長引くことがあります。この原因は複雑で可変である可能性があります。誰が作業すべきかが明確ではないことが原因である場合もあります。場合によっては、問題にはコードベースの特定の部分に関する調査や専門知識が必要で、作業に最適な候補は他の問題でビジー状態です。多くの場合、他の緊急作業は最初に対応する必要があります。これらの原因のいずれかまたはすべてにより、問題が解決されない可能性があります。 には、Amazon Q と呼ばれる生成 AI アシスタントとの統合 CodeCatalyst が含まれており、タイトルとその説明に基づいて問題を分析できます。問題を Amazon Q に割り当てると、評価用のドラフトソリューションの作成が試みられます。これにより、Amazon Q はすぐに対処すべきリソースがない問題に対するソリューションに取り組みながら、お客様とチームが注意が必要な問題に集中して作業を最適化できます。

> **Note**
>
> Amazon Bedrock を搭載： AWS 自動不正使用検出を実装。ソフトウェア開発用の Amazon Q デベロッパーエージェントによる Amazon Q 機能への問題の割り当ては Amazon Bedrock 上に構築されているため、ユーザーは Amazon Bedrock に実装されているコントロールを最大限に活用して、安全性、セキュリティ、人工知能 (AI) の責任ある使用を強制できます。

Amazon Q は、単純な問題と単純な問題に最適です。最良の結果を得るには、プレーン言語を使用して、実行したいことを明確に説明してください。以下は、Amazon Q が機能するように最適化された問題の作成に役立つベストプラクティスです。

> **Important**
>
> 生成 AI 機能は、米国西部 (オレゴン) リージョンでのみ使用できます。

- シンプルにします。Amazon Q は、問題のタイトルと説明で説明できる簡単なコード変更と修正に最適です。あいまいなタイトルや過度に花のような説明や矛盾する説明に関する問題は割り当てないでください。
- 具体的にします。問題の解決に必要な正確な変更についてより多くの情報を提供できるほど、Amazon Q が問題を解決するソリューションを作成できる可能性が高くなります。可能な場合は、変更する APIs の名前、更新するメソッド、変更が必要なテスト、その他考えられる詳細など、特定の詳細を含めます。
- Amazon Q に割り当てる前に、問題のタイトルと説明にすべての詳細が含まれていることを確認してください。 Amazon Q に割り当てた後は、問題のタイトルや説明を変更することはできません。そのため、Amazon Q に割り当てる前に、問題に必要なすべての情報が揃っていることを確認してください。
- 1 つのソースリポジトリ でコード変更が必要な問題のみ割り当てます。Amazon Q は、 の 1 つのソースリポジトリのコードでのみ動作します CodeCatalyst。リンクされたリポジトリはサポートされていません。Amazon Q に割り当てる前に、問題が 1 つのソースリポジトリの変更のみを必要とすることを確認してください。
- 各ステップ を承認するために、Amazon Q が提案するデフォルト を使用します。デフォルトでは、Amazon Q は実行するステップごとに承認を必要とします。これにより、問題に関するコメントだけでなく、Amazon Q が作成するプルリクエストでも Amazon Q とやり取りできます。これにより、Amazon Q とのよりインタラクティブなエクスペリエンスが提供され、アプローチを調整し、問題を解決するために作成するコードを絞り込むのに役立ちます。

> **Note**
>
> Amazon Q は、問題やプルリクエストの個々のコメントには応答しませんが、アプローチを再検討したり、リビジョンを作成したりするように求められたときにレビューされます。

- Amazon Q が提案するアプローチを常に慎重に確認してください。 そのアプローチを承認すると、Amazon Q はそのアプローチに基づいてコードの生成作業を開始します。続行するように Amazon Q に指示する前に、アプローチが正しいように見え、予想されるすべての詳細が含まれていることを確認してください。
- のレビュー前に、ワークフローをデプロイする可能性のある既存のワークフローがない場合にのみ、Amazon Q によるワークフローの作業を許可してください。プロジェクトには、プルリクエストイベントの実行を開始するようにワークフローが設定されている場合があります。その場合、ワークフロー YAML の作成または更新を含む Amazon Q が作成するプルリクエストは、プルリクエストに含まれるワークフローの実行を開始する可能性があります。ベストプラクティスとして、作成したプルリクエストを確認して承認する前に、これらのワークフローを自動的に実行するワークフローがプロジェクトにないことを確認した場合を除き、Amazon Q によるワークフローファイルの作業を許可しないでください。

詳細については、チュートリアル: CodeCatalyst 生成 AI 機能を使用して開発作業を高速化する「」および「生成 AI 機能の管理」を参照してください。

問題の見積もり

アジャイル開発では、見積もりはストーリーポイントと呼ばれます。問題のあいまいさと複雑さに加えて、問題の見積もりを使用して、必要な作業量を表すことができます。リスク、難易度、不明度が高い問題については、より高い見積りを使用することを検討してください。

問題の見積もりを開始する前に、まずプロジェクトに使用する見積りのタイプを選択する必要があります。デフォルトでは、2 つのタイプから選択できます。T ウェアのサイズ設定またはフィボナッチシーケンスを効果的に使用するには、チームが各サイズが表す内容に合わせる必要があります。各見積りが表すものをまとめて、それらの見積りを各問題に適用し始めます。定期的なレビューを検討する

以下の手順に従って、 の問題の労力推定の設定を行います CodeCatalyst。

問題の労力推定を設定するには

1. ナビゲーションペインで、問題 を選択します。
2. 「アクティブな問題」を選択して問題ビューのスイッチャードロップダウンメニューを開き、「設定」を選択します。
3. 「基本設定」セクションの「推定」で、推定値の表示方法を選択します。利用可能な見積りのタイプは、T ウェアのサイズ設定、フィボナッチシーケンシング、または見積りの非表示です。

プロジェクトの推定設定が見積りを非表示 に設定されている場合、プロジェクトの問題には見積りフィールドがありません。

推定タイプが更新されると、データは失われず、すべての問題の推定値が自動的に変換されます。変換マッピングを次の表に示します。

T バッグのサイズ	フィボナッチシーケンス
XS	1
XS	2
S	3
M	5
L	8
XL	13

問題の見積りを追加または変更するには、問題 を編集します。

での問題の編集とコラボレーション CodeCatalyst

目次

問題の編集

問題のタイトル、説明、ステータス、担当者、優先度、見積り、またはラベルを編集するには、次の手順に従います。

問題を編集するには

1. 編集する問題を選択して、問題の詳細を表示します。問題を見つける方法については、「」を参照してください問題の検出と表示。
2. 問題タイトルを編集するには、タイトルを選択し、新しいタイトルを入力して、Enter キーを押します。
3. 説明を編集するには、説明を選択し、新しい説明を入力し、Enter キーを押します。Markdown を使用して書式を追加できます。
4. タスク では、問題のタスクを表示および管理できます。詳細については、「問題に関するタスクの管理」を参照してください。
5. ステータス 、見積り 、または優先度 を編集するには、それぞれのドロップダウンメニューからオプションを選択します。
6. ラベル では、既存のラベルの追加、新しいラベルの作成、またはラベルの削除を行うことができます。

   a. 既存のラベルを追加するには、 + ラベルを追加 を選択し、リストからラベルを選択します。フィールドに検索語を入力すると、その語句を含むすべてのラベルをプロジェクトで検索できます。

   b. 新しいラベルを作成して追加するには、 + ラベルを追加 検索フィールドに作成するラベルの名前を入力し、Enter キーを押します。

   c. ラベルを削除するには、削除するラベルの横にある X アイコンを選択します。すべての問題からラベルを削除すると、ラベルは問題設定 のラベルセクションの未使用ラベルセクションに表示されます。フィルターを使用したり、問題にラベルを追加したりすると、未使用のラベルがラベルのリストの末尾に表示されます。すべてのラベル (使用済みと未使用) の概要と、ラベルを含む問題は、問題設定 で確認できます。
7. 問題を割り当てるには、「担当者」セクションで「 + 担当者を追加」を選択し、リストから担当者を検索して選択します。+ 追加を選択すると、自分を担当者としてすばやく追加できます。
8. 添付ファイル では、添付ファイルを追加、ダウンロード、または削除できます。詳細については、「添付ファイルの使用」を参照してください。
9. プルリクエストをリンクするには、プルリクエストのリンク を選択し、リストからプルリクエストを選択するか、URL または ID を入力します。プルリクエストのリンクを解除するには、リンク解除アイコンを選択します。

> **Tip**
>
> 問題へのプルリクエストへのリンクを追加したら、リンクされたプルリクエストのリストでその ID を選択することで、そのプルリクエストにすばやく移動できます。プルリクエストの URL を使用して、問題ボードとは異なるプロジェクトにあるプルリクエストをリンクできますが、そのプロジェクトのメンバーであるユーザーのみがそのプルリクエストを表示またはナビゲートできます。

10. (オプション) 既存のカスタムフィールドの追加と設定、新しいカスタムフィールドの作成、またはカスタムフィールドの削除を行います。問題には、複数のカスタムフィールドを含めることができます。

    a. 既存のカスタムフィールドを追加するには、リストからカスタムフィールドを選択します。フィールドに検索語を入力すると、プロジェクト内のその語を含むすべてのカスタムフィールドを検索できます。

    b. 新しいカスタムフィールドを作成して追加するには、作成するカスタムフィールドの名前を検索フィールドに入力します。次に、作成するカスタムフィールドのタイプを選択し、値を設定します。

    c. カスタムフィールドを削除するには、削除するカスタムフィールドの横にある X アイコンを選択します。すべての問題からカスタムフィールドを削除すると、カスタムフィールドは削除され、フィルタリング時に表示されなくなります。

添付ファイルの使用

の問題に添付ファイルを追加して CodeCatalyst 、関連ファイルへのアクセスを容易にすることができます。問題の添付ファイルを管理するには、次の手順に従います。

問題に追加される添付ファイルのサイズは、スペースのストレージクォータにカウントされます。プロジェクトの添付ファイルの表示と管理については、「」を参照してください添付ファイルの表示と管理。

> **Important**
>
> 問題への添付ファイルは、Amazon によってスキャンまたは分析されません CodeCatalyst。どのユーザーも、悪意のあるコードやコンテンツが含まれている可能性のある問題に添付ファイルを追加することができます。添付ファイルの管理や悪意のあるコード、コンテン

ツ、ウイルスからの保護に関しては、ユーザーがベストプラクティスを認識していることを確認してください。

添付ファイルを追加、ダウンロード、または削除するには

1. 添付ファイルを管理する問題を選択します。問題を見つける方法については、「」を参照してください問題の検出と表示。
2. 添付ファイルを追加するには、ファイルのアップロード を選択します。オペレーティングシステムのファイルエクスプローラーの ファイルに移動し、選択します。Open を選択して添付ファイルとして追加します。最大アタッチメントサイズなどのクォータ情報については、「」を参照してくださいの問題のクォータ CodeCatalyst。

   添付ファイルの名前とコンテンツタイプには、次の制限があることに注意してください。

   - ファイル名では、次の文字は使用できません。
     - コントロール文字: `0x00-0x1f` および `0x80-0x9f`
     - 予約文字: /、?、<、>、\、:、*、|、および "
     - Unix 予約ファイル名: `.`および `..`
     - 末尾のピリオドとスペース
     - Windows 予約ファイル名: `CON`, `PRN`, `AUX`, `NUL`, `COM1`, `COM2`, `COM3`, `COM4`, `COM5`, `COM6`, `COM7`, `COM8`, `COM9`, `LPT1`, `LPT2`, `LPT3`, `LPT4`, `LPT5`, `LPT6`, `LPT7`, `LPT8`, and `LPT9`
   - アタッチメントのコンテンツタイプは、次のメディアタイプパターンに従う必要があります。

     ```
     media-type = type "/" [tree "."] subtype ["+" suffix]* [";" parameter];
     ```

     例えば `text/html; charset=UTF-8` です。
3. 添付ファイルをダウンロードするには、ダウンロードする添付ファイルの横にある省略記号メニューを選択し、ダウンロードを選択します。
4. 添付ファイルの URL をコピーするには、URL をコピーする添付ファイルの横にある省略記号メニューを選択し、URL のコピー を選択します。
5. 添付ファイルを削除するには、削除する添付ファイルの横にある省略記号メニューを選択し、「削除」を選択します。

添付ファイルの表示と管理

プロジェクトの問題に追加されたすべての添付ファイルを含むテーブルを、問題設定で表示できます。この表には、コンテンツタイプ、追加日時、追加した問題とそのステータス、ファイルサイズなどの各添付ファイルの詳細が含まれています。

このテーブルを使用すると、完了またはアーカイブされた問題に関する大きな添付ファイルを簡単に識別して、スペースストレージを解放できます。

> **Important**
>
> 問題への添付ファイルは、Amazon によってスキャンまたは分析されません CodeCatalyst。どのユーザーも、悪意のあるコードやコンテンツが含まれている可能性のある問題に添付ファイルを追加することができます。添付ファイルの管理や悪意のあるコード、コンテンツ、ウイルスからの保護に関しては、ユーザーがベストプラクティスを認識していることを確認してください。

プロジェクト内のすべての問題添付ファイルを表示および管理するには

1. ナビゲーションペインで、問題 を選択します。
2. 省略記号アイコンを選択し、設定 を選択します。
3. 添付ファイルタブを選択します。

問題に関するタスクの管理

タスクを問題に追加して、その問題の作業をさらに分類、整理、追跡できます。

問題のタスクを管理するには

1. タスクを管理する問題を選択します。問題を見つける方法については、「」を参照してください問題の検出と表示。
2. タスク では、問題のタスクを表示および管理できます。
    1. タスクを追加するには、テキストフィールドにタスク名を入力し、Enter キーを押します。
    2. タスクを完了としてマークするには、タスクのチェックボックスをオンにします。
    3. タスクの詳細を表示または更新するには、リストから選択します。

4. タスクの順序を変更するには、チェックボックスの左側からタスクを選択してドラッグします。
5. タスクを削除するには、タスクの省略記号メニューを選択し、 の削除を選択します。

問題をブロック済みまたはブロック解除済みとしてマークする

何かが原因で問題に対処できない場合は、ブロックされた としてマークすることをお勧めします。例えば、まだマージされていないコードベースの別の部分への変更に依存している場合、問題がブロックされる可能性があります。

問題をブロックとしてマークすると、 は赤色のブロック済みラベルを問題 CodeCatalyst に追加し、バックログやアーカイブ、またはボードで非常によく見えるようにします。

外部の状況が解決された場合は、問題のブロックを解除できます。

問題をブロック済みとしてマークするには

1. ブロックとしてマークする問題を開きます。問題を見つける方法については、「」を参照してください問題の検出と表示。
2. アクション を選択し、ブロックされた としてマーク を選択します。

問題のブロックを解除するには

1. ブロックを解除する問題を開きます。問題を見つける方法については、「」を参照してください問題の検出と表示。
2. アクション を選択し、ブロック解除された としてマーク を選択します。

コメントの追加、編集、削除

問題についてコメントを残すことができます。コメントでは、他のスペースメンバー、スペース内の他のプロジェクト、関連する問題、およびコードにタグ付けできます。

問題にコメントを追加するには

1. プロジェクトに移動します。
2. ナビゲーションバーで問題 を選択します。

3. コメントを追加する問題を選択します。問題を見つける方法については、「」を参照してください問題の検出と表示。
4. コメントフィールドにコメントを入力します。Markdown を使用して書式を追加できます。
5. [送信] を選択します。

コメントを編集するには

問題に対して行ったコメントを編集できます。編集できるのは、作成したコメントのみです。

1. プロジェクトに移動します。
2. ナビゲーションバーで問題 を選択します。
3. コメントを編集する問題を選択します。問題を見つける方法については、「」を参照してください問題の検出と表示。
4. コメントを編集するには、編集するコメントを見つけます。

> **Tip**
>
> コメントは、最も古いものまたは最新のものに基づいて最初にソートできます。コメントは一度に 10 個ロードされます。

5. 省略記号アイコンを選択し、編集 を選択します。
6. コメントを編集します。Markdown を使用して書式を追加できます。
7. [保存] を選択します。コメントが更新されました。

コメントを削除するには

問題に対して行ったコメントは削除できます。削除できるのは、作成したコメントのみです。コメントを削除すると、ユーザー名が表示されますが、このコメントは元のコメントテキストの代わりに削除されます。

1. プロジェクトに移動します。
2. ナビゲーションバーで問題 を選択します。
3. コメントを削除する問題を選択します。問題を見つける方法については、「」を参照してください問題の検出と表示。
4. 省略記号アイコンを選択し、 の削除 を選択し、確認 を選択します。

コメントでのメンションの使用

スペースメンバー、スペース内の他のプロジェクト、関連する問題、およびコードをコメントで言及できます。これにより、言及したユーザーまたはリソースへのクイックリンクが作成されます。

コメントで @mention するには

1. プロジェクトに移動します。
2. ナビゲーションバーで問題 を選択します。
3. 編集する問題を選択して、問題の詳細を表示します。問題を見つける方法については、「」を参照してください問題の検出と表示。
4. コメントの追加テキストボックスを選択します。
5. 別のユーザーについて言及@*user_name*するには、「」と入力します。
6. プロジェクトについて言及@*project_name*するには、「」と入力します。
7. 別の問題について言及@*issue_number*するには、 @*issue_name*または を入力します。
8. と入力@*file_name*すると、ソースリポジトリ内の特定のファイルまたはコードに言及できます。

   > **Note**
   >
   > に一致する用語を含む上位 5 つの項目 (ユーザー、ソースリポジトリ、プロジェクトなど) のリストは、入力時に入力@mentionされます。

9. 言及したい項目を選択します。項目がどこにあるかを示すパスがコメントテキストボックスに入力されます。
10. コメントを終了し、送信 を選択します。

問題の検出と表示

以下のセクションでは、 CodeCatalyst プロジェクト内の問題を効果的に検索して表示する方法について説明します。

問題の検索

特定のパラメータを検索することで問題を見つけることができます。検索の改良の詳細については、「」を参照してくださいでコード、問題、プロジェクト、ユーザーを検索する CodeCatalyst。

問題を検索するには

1. プロジェクトに移動します。
2. 検索バーを使用して、問題または問題に関連する情報を検索します。クエリパラメータを使用して検索を絞り込むことができます。詳細については、「でコード、問題、プロジェクト、ユーザーを検索する CodeCatalyst」を参照してください。

ソートの問題

デフォルトでは、 の問題 CodeCatalyst は手動順序 でソートされます。手動順序では、ユーザーが移動した順序で問題が表示されます。手動順にソートすると、問題をドラッグアンドドロップして順序を変更できます。このソートオプションは、問題のバックログを整理し、問題に優先順位を付けるのに役立ちます。

次の表は、グリッドビューとボードビューの両方で問題をどのようにソートできるかを示しています。

グリッドビューのソートオプション	ボードビューのソートオプション
手動注文	手動注文
最終更新日	最終更新日
優先度	優先度
見積り	見積り
タイトル	タイトル
ID	
ステータス	
ブロック	
カスタム フィールド	

次の手順を使用して、問題のソート方法を変更します。

問題をソートするには

1. プロジェクトに移動します。
2. ナビゲーションペインで、問題 を選択します。デフォルトのビューはボード です。
3. （オプション)「アクティブな問題」を選択して、問題ビューのスイッチャードロップダウンメニューを開き、別の問題ビューに移動します。
4. グリッドビューをソートするには、次の 2 つのオプションがあります。
   a. ソートするフィールドのヘッダーを選択します。ヘッダーを選択すると、昇順と降順が切り替わります。
   b. ソート基準ドロップダウンメニューを選択し、ソート基準にするパラメータを選択します。問題は昇順でソートされます。
5. ボードビューをソートするには、ソート基準ドロップダウンメニューを選択し、ソート基準となるパラメータを選択します。問題は昇順でソートされます。

問題のグループ化

グループ化は、担当者、ラベル、優先度などの複数のパラメータによってボード上の問題を整理するために使用されます。

問題をグループ化するには

1. プロジェクトに移動します。
2. ナビゲーションペインで、問題 を選択します。デフォルトのビューはボード です。
3. （オプション) アクティブな問題を選択して、問題ビューのスイッチャードロップダウンメニューを開き、別の問題ビューに移動します。
4. グループ を選択します。
5. Group by で、グループ化するパラメータを選択します。
   - 担当者または優先度 を選択した場合は、グループの順序 を選択します。
   - ラベル を選択した場合は、ラベルを選択し、グループ順序 を選択します。
6. （オプション) 空のグループを表示トグルを選択して、現在問題が割り当てられていないグループを表示または非表示にします。
7. 選択すると、ビューが更新されます。問題は、設定されたパラメータに一致するグループにのみ表示されます。

問題のフィルタリング

フィルタリングを使用して、指定された名前、優先度、ラベル、カスタムフィールド、または担当者を含む問題を検索します。

問題をフィルタリングするには

1. プロジェクトに移動します。
2. ナビゲーションペインで、問題 を選択します。
3. （オプション）「アクティブな問題」を選択して、問題ビューのスイッチャードロップダウンメニューを開き、別の問題ビューに移動します。

> **Note**
>
> 問題名または説明の文字列に基づいてフィルタリングするには、問題検索バーに文字列を入力します。

4. フィルター を選択し、+ フィルターを追加 を選択します。
5. フィルタリングするパラメータを選択します。複数のフィルターとパラメータを選択できます。フィルターを設定して、すべてのフィルターまたは個々のフィルターに一致する問題を表示するには、 および または または を選択します。ビューが更新され、フィルターに一致する問題が表示されます。

問題の進行

すべての問題にはライフサイクルがあります。では CodeCatalyst、問題は通常、バックログのドラフトとして始まります。その問題の作業を開始すると、別のステータスカテゴリに移動され、完了するまでさまざまなステータスに移行し、アーカイブされます。次の方法で、ライフサイクルを通じて問題を移動または進行できます。

- バックログとボードの間で問題を移動できます。
- 進行中の問題は、さまざまな完了段階に移行できます。
- 完了した問題をアーカイブできます。

バックログとボード間の問題の移動

問題の処理を開始すると、バックログからボードに問題を移動できます。作業が延期された場合は、問題をバックログに戻すこともできます。

バックログとボード間で問題を移動するには

1. プロジェクトに移動します。
2. ナビゲーションペインで、問題 を選択します。デフォルトのビューはボード です。
3. 問題をボードからバックログに移動するには：
   a. 移動する問題を選択します。問題を見つける方法については、「」を参照してください問題の検出と表示。
   b. ステータスメニューからバックログを選択します。
4. 問題をバックログからボードに移動するには：
   a. バックログに移動するには、ボード を選択し、バックログ を選択します。
   b. 移動する問題を選択します。問題を見つける方法については、「」を参照してください問題の検出と表示。
   c. ボードに追加 を選択するか、バックログ 以外のステータスを選択します。

ボードのライフサイクルステージを通じて問題を進行する

ボード内で問題は、完了するまで異なるステータスで移動できます。

ボード内で問題を移動するには

1. ナビゲーションペインで、問題 を選択します。デフォルトのビューはボード です。
2. 次のいずれかを行います。
   - 問題を別のステータスにドラッグアンドドロップします。
   - 問題を選択し、ステータスドロップダウンメニューからステータスを選択します。
   - 問題を選択し、次へ を選択します。

問題のアーカイブについては、「」を参照してください問題のアーカイブ。

グループ間の問題の移動

すべての問題ビューとボードビューで、さまざまなパラメータ別に問題をグループ化できます。 問題がグループ化されている場合は、あるグループから別のグループへ問題を移動できます。あるグループから別のグループに問題を移動すると、ターゲットグループと一致するように問題がグループ化されているフィールドが自動的に編集されます。

シナリオの例として、Wang Xiulan と Saanvi Sarkar という 2 人の問題 CodeCatalyst が割り当てられている を使用している会社があるとします。ボードは ごとにグループ化されAssignee、割り当て先ごとに 1 つずつ 2 つのグループがあります。Wang Xiulan グループから Saanvi Sarkar グループに問題を移動すると、問題の担当者を Saanvi Sarkar に更新します。

問題のアーカイブ

> **Note**
>
> 問題はプロジェクト内で削除されず、アーカイブされます。問題を削除するには、プロジェクトを削除する必要があります。

プロジェクトで不要になった問題はアーカイブできます。問題をアーカイブすると、アーカイブされた問題を除外するすべてのビューから によって CodeCatalyst 削除されます。アーカイブされた問題は、アーカイブされた問題のデフォルトビューで表示でき、必要に応じてアーカイブ解除できます。

次の場合に問題をアーカイブします。

- 問題は完了し、完了 列で不要になりました。
- 作業する予定はありません。
- 誤って作成しました。
- アクティブな問題の最大数に達しました。

問題をアーカイブするには

1. アーカイブする問題を開きます。問題を見つける方法については、「」を参照してください問題の検出と表示。
2. アクション を選択し、アーカイブに移動 を選択します。
3. (オプション) 完了ステータスの複数の問題をすばやくアーカイブするには、ボード上の完了ステータスの上部にある縦の省略記号を選択し、アーカイブの問題を選択します。

問題をアーカイブ解除するには

1. アーカイブ解除する問題を開きます。アーカイブされた問題のリストを表示するには、問題ビューのスイッチャードロップダウンメニューからアーカイブされた問題ビューを開きます。問題を見つける方法については、「」を参照してください問題の検出と表示。
2. Unarchive を選択します。

エクスポートに関する問題

現在のビューの問題を .xlsx ファイルにエクスポートできます。問題をエクスポートするには、次のステップを実行します。

問題をエクスポートするには

1. プロジェクトに移動します。
2. ナビゲーションバーで問題 を選択します。
3. 「アクティブな問題」を選択して、問題ビューのスイッチャードロップダウンメニューを開き、エクスポートする問題を含むビューに移動します。ビューに表示される問題のみがエクスポートされます。
4. 省略記号メニューを選択し、Excel にエクスポート を選択します。
5. .xlsx ファイルをダウンロードします。デフォルトでは、プロジェクトの名前とエクスポートが完了した日付というタイトルが付けられます。

バックログ、ラベル、ボードでの作業の整理

すべてのチームが同じように作業するわけではありません。Amazon で問題がどのように表示され、割り当てられるかを設定 CodeCatalyst して、作業内容とその作業のステータスを正確に把握できます。ユーザー全員が同じ推定方法を使用できるように、問題を許可する推定方法を選択できます。カスタムラベルとステータスを作成して、作業のビューをフィルタリングするためにも使用できます。チームの仕組みに応じて、1 つの問題に対して複数の担当者を許可するか、1 人のユーザーにのみ問題を割り当てるかを設定できます。また、問題のカスタムビューを作成して、自分またはチームに最も関連性の高い情報を表示する方法で作業を表示することもできます。

ラベルを使用した作業の分類

問題のラベルをカスタマイズできます。これには、ラベルの編集と色の変更が含まれます。ラベルは、作業の分類と整理に役立ちます。例えば、ソフトウェアの特定の側面、または異なるグループやチームに対してラベルを作成できます。

トピック

ラベルの作成

では CodeCatalyst、新しい問題を作成するとき、または既存の問題を編集するときにラベルを追加することでラベルを作成します。詳細については、「での問題の作成 CodeCatalyst」および「での問題の編集とコラボレーション CodeCatalyst」を参照してください。

ラベルの編集

既存のラベルの名前または色を変更するには、次の手順に従います。

ラベルを編集するには

1. ナビゲーションペインで、問題 を選択します。
2. 「アクティブな問題」を選択して問題ビューのスイッチャードロップダウンメニューを開き、「設定」を選択します。
3. ラベルタイルには、プロジェクトで使用されるラベルのリストが表示されます。編集するラベルの横にある編集アイコンを選択します。次の 1 つ以上の操作を行います。
   a. ラベルの名前を編集します。
   b. 色を変更するには、色ホイールを選択します。ピッカーを使用して新しい色を選択します。
4. ラベルに加えた変更を保存するには、チェックマークアイコンを選択します。
5. 変更されたラベルが、使用可能なラベルのリストに表示されるようになりました。また、そのラベルを使用している問題の数も確認できます。

> **Note**
>
> 各ラベルの横に表示される番号を選択すると、「すべての問題」ページに移動し、そのラベルを含むすべての問題を表示できます。

ラベルの削除

現在、 で問題ラベルを削除することはできません CodeCatalyst。すべての問題からラベルを削除すると、ラベルは問題設定 のラベルセクションの未使用ラベルセクションに表示されます。フィルターを使用したり、問題にラベルを追加したりすると、未使用のラベルがラベルのリストの末尾に表示されます。すべてのラベル (使用済みおよび未使用) の概要と、ラベルを含む問題は、問題設定 で確認できます。

カスタムフィールドでの作業の整理

カスタムフィールドを作成して、プロジェクトの作業を整理および表示できます。カスタムフィールドは、フィルターで使用可能なフィルターのリストに追加されるため、カスタムフィールドで問題をフィルタリングできます。カスタムフィールドは名前と値のペアです。カスタムフィールドの名前でフィルタリングし、次にそのカスタムフィールドの値でフィルタリングします。

問題には、複数のカスタムフィールドが含まれる場合があります。

カスタムフィールドの作成

では CodeCatalyst、カスタムフィールドは、問題の作成時または既存の問題の編集時に追加して作成します。詳細については、「での問題の作成 CodeCatalyst」および「での問題の編集とコラボレーション CodeCatalyst」を参照してください。

カスタムフィールドの削除

カスタムフィールドを削除するには、追加される各問題からカスタムフィールドを削除する必要があります。カスタムフィールドが削除されると、フィルター にカスタムフィールドが表示されなくなります。フィルターを使用して、カスタムフィールドのすべての問題を表示し、問題を編集して削除できます。詳細については、「問題の検出と表示」および「問題の編集」を参照してください。

カスタムステータスの追跡作業

ボードにカスタムステータスを追加できます。各カスタムステータスは、ドラフト 、未開始 、開始 、完了 のいずれかのカテゴリに属している必要があります。ステータスカテゴリは、ステータスの整理やデフォルトビューの入力に役立ちます。ステータスとステータスカテゴリの詳細については、ステータスとステータスカテゴリ「」を参照してください。ビューの詳細については、「」を参照してください問題の検出と表示。

ステータスを作成するには

1. ナビゲーションペインで、問題 を選択します。
2. 「アクティブな問題」を選択して問題ビューのスイッチャードロップダウンメニューを開き、「設定」を選択します。
3. ステータス で、ステータスを表示するカテゴリの横にあるプラスアイコンを選択します。
4. ステータスに名前を付け、チェックマークアイコンを選択します。

> **Note**
>
> ステータスの追加をキャンセルするには、X アイコンを選択します。

カスタムステータスがボードに表示され、問題を作成するときにオプションとして が表示されます。

ステータスを編集するには

1. ナビゲーションペインで、問題 を選択します。
2. 「アクティブな問題」を選択して問題ビューのスイッチャードロップダウンメニューを開き、「設定」を選択します。
3. ステータス で、編集または変更するステータスの横にある編集アイコンを選択します。
4. ステータスを編集し、チェックマークアイコンを選択します。

これで、編集されたステータスがボードに表示されます。

ステータスを移動するには

1. ナビゲーションペインで、問題 を選択します。
2. 省略記号アイコンを選択し、設定 を選択します。
3. ステータス で、移動するステータスを選択します。
4. ステータスをドラッグアンドドロップします。

> **Note**
>
> ステータスを移動できるのは、指定されたカテゴリ内のみです。

これで、ボードのステータスが並べ替えられます。

ステータスを無効にするには

1. ナビゲーションペインで、問題 を選択します。
2. 「アクティブな問題」を選択して問題ビューのスイッチャードロップダウンメニューを開き、「設定」を選択します。
3. ステータス で、非アクティブ化するステータスを選択します。
4. 非アクティブ化するステータスで、ステータスのトグルを選択します。ステータスがグレー表示になりました。

> **Note**
>
> 非アクティブ化されたステータスは、すべての問題がボードから移動されるまでボードに表示されます。非アクティブ化されたステータスに問題を追加することはできません。

5. 非アクティブ化されたステータスを再アクティブ化するには、ステータスのトグルを選択します。ステータスはグレー表示されなくなりました。

> **Note**
>
> 各カテゴリには少なくとも 1 つのアクティブなステータスが必要です。カテゴリにステータスが 1 つしかない場合は、非アクティブ化できません。

問題労力の見積もりの設定

以下の手順に従って、 の問題の労力推定の設定を行います CodeCatalyst。

問題の労力推定を設定するには

1. ナビゲーションペインで、問題 を選択します。
2. 「アクティブな問題」を選択して問題ビューのスイッチャードロップダウンメニューを開き、「設定」を選択します。
3. 「基本設定」セクションの「推定」で、推定値の表示方法を選択します。利用可能な見積りのタイプは、T ウェアのサイズ設定、フィボナッチシーケンシング、または Hide の見積りです。推定タイプが更新されると、データは失われず、すべての問題の推定値が自動的に変換されます。変換マッピングを次の表に示します。

T バッグのサイズ	フィボナッチシーケンス
XS	1
XS	2
S	3
M	5
L	8
XL	13

複数の担当者の有効化または無効化

以下の手順に従って、 の問題に対して複数の担当者の設定を行います CodeCatalyst。

複数の担当者を有効または無効にするには

1. ナビゲーションペインで、問題 を選択します。
2. 「アクティブな問題」を選択して問題ビューのスイッチャードロップダウンメニューを開き、「設定」を選択します。

3. 「基本設定」セクションの「担当者」で、インジケータを切り替えて、複数の担当者を同じ問題に割り当てることができるようにします。問題には、最大 10 人の担当者が含まれる場合があります。このオプションを有効にしない場合、問題に割り当てることができるのは 1 人の担当者のみです。

問題ビューの作成

ビューを作成して、特定のフィルターセットに一致する問題をすばやく表示できます。これにより、時間を節約し、以前にフィルタリング、グループ化、またはソートした問題をすばやく表示できます。

問題ビューを作成するには

1. ナビゲーションペインで、問題 を選択します。
2. (オプション) ユースケースによっては、既存のビューからビューを作成することもできます。別のビューに移動するには、「アクティブな問題」を選択して問題ビューのスイッチャードロップダウンメニューを開き、ビューを選択します。
3. (オプション) ビューを作成する前に、フィルター、グループ化、ソートを設定します。これらはビューの作成時に追加できますが、以前に追加した場合は、ビューに表示される内容をプレビューしてから作成できます。
4. ヘッダーバーから問題ビューのスイッチャードロップダウンメニューを開きます。ステータスに基づいて問題が列に表示されるボードビューを作成するには、ボード列で + を選択します。問題がリストに表示されるグリッドビューを作成するには、グリッド列で + を選択します。ビューのタイプは、作成前に変更することができます。
5. ビューの作成ダイアログボックスに、ビューの名前を入力します。
6. フィルター 、 による問題をグループ化、フィールドによる問題をソート は、現在のビューの設定に基づいて入力されます。必要に応じて更新します。
7. ビューの作成 を選択してビューを作成し、切り替えます。

の問題のクォータ CodeCatalyst

次の表に、Amazon の問題のクォータと制限を示します CodeCatalyst。Amazon のクォータの詳細については、 CodeCatalyst「」を参照してくださいのクォータ CodeCatalyst。

リソース	デフォルトのクォータ
アクティブな問題	プロジェクトあたり最大 1,000。
アタッチメントサイズ	アタッチメントあたり最大 500MB。 アタッチメントストレージの最大合計は、スペース全体のストレージ制限の影響を受けます。詳細については、「 の料金」を参照してください。
問題の合計数 (アクティブおよびアーカイブ済み)	プロジェクトあたり最大 100,000。
保存されたビュー	プロジェクトごとに最大 50 個の保存された問題ビュー。
問題にリンクできるプルリクエストの数	問題ごとに最大 50 のプルリクエスト。
ステータス (プロジェクトごと)	プロジェクトあたり最大 50 個。
ステータス (問題ごと)	問題あたり最大 50 個。
ラベル (プロジェクトごと)	プロジェクトあたり最大 200。
ラベル (問題ごと)	問題あたり最大 50 個。
カスタムフィールド (問題ごと)	問題あたり最大 50 個。
担当者	問題ごとに最大 10 個。
コメント	問題あたり最大 1,000。
タスク	問題あたり最大 100。

で ID、アクセス許可、アクセスを設定する CodeCatalyst

Amazon に CodeCatalyst 初めてサインインするときは、ビルダー ID AWS を作成します。 AWS ビルダー IDs には存在しません AWS Identity and Access Management。最初のサインイン時に選択したユーザー名は、ID の一意のユーザー ID になります。

では CodeCatalyst、次の 2 つの方法のいずれかで初めてサインインできます。

- スペースの作成の一環として。
- のプロジェクトまたはスペースへの招待を受け入れる一環です CodeCatalyst。

ID に関連付けられたロールによって、 で実行できるアクションが決まります CodeCatalyst。プロジェクト管理者 や寄稿者 などのプロジェクトロールはプロジェクトに固有であるため、あるプロジェクトでは 1 つのロールを、別のプロジェクトでは別のロールを持つことができます。スペースを作成すると、 CodeCatalyst によって自動的にスペース管理者ロールが割り当てられます。ユーザーがプロジェクトへの招待を受け入れると、 はそれらの ID をスペース CodeCatalyst に追加し、制限付きアクセスロールを割り当てます。プロジェクトにユーザーを招待するときは、プロジェクトに含めるロールを選択します。これにより、プロジェクト内で実行できるアクションとできないアクションが決まります。プロジェクトで作業するほとんどのユーザーは、タスクを実行するために Contributor ロールのみが必要です。詳細については、「ユーザーロールによるアクセス許可の付与」を参照してください。

プロジェクトロールに加えて、プロジェクト内のユーザーは、Git クライアントまたは統合開発環境 (IDEsが必要です。プロジェクトメンバーは、アイデンティティに関連付けられたアプリケーション固有のパスワードとして、この PAT をサードパーティーアプリケーションで使用できます CodeCatalyst 。例えば、ソースリポジトリのクローンをローカルコンピュータに作成する場合は、PAT と CodeCatalyst ユーザー名を指定する必要があります。

ワークフローにアクションをデプロイするときに、サービスロールを使用して AWS CloudFormation スタック CodeCatalyst や AWS リソースへのアクセスなどのアクションを実行することで、 とリソース間のアクセスを設定できます。プロジェクトテンプレートに含まれるワークフローアクションを実行するには、 CodeCatalyst と AWS リソース間のアクセスを設定する必要があります。

トピック

- ユーザーロールによるアクセス許可の付与
- 個人用アクセストークンを使用してリポジトリアクセスをユーザーに付与する

- 個人接続による GitHub リソースへのアクセス
- 多要素認証 (MFA) でサインインするように AWS Builder ID を設定する
- Amazon のセキュリティ CodeCatalyst
- ログ記録を使用したイベントと API コールのモニタリング
- での ID、アクセス許可、アクセスのクォータ CodeCatalyst
- トラブルシューティング

ユーザーロールによるアクセス許可の付与

Amazon では CodeCatalyst、プロジェクトレベルとスペースレベルの両方でユーザーにロールを割り当てることができます。プロジェクトでは、ロールは、ユーザーがそのプロジェクトのリソースを使用してプロジェクトで実行できる操作を指定します。ユーザーは、プロジェクトに参加するとスペースのメンバーシップを取得します。ユーザーをスペースの管理者として追加または削除できます。Space 管理者ロールには、 の任意のロールの最も広範なアクセス許可があります CodeCatalyst。ベストプラクティスとして、ジョブの実行に必要な最小限のアクセス許可をユーザーに割り当てます。

スペース内のユーザーにロールを割り当てることができます。また、メンバーであるプロジェクトのユーザーにロールを割り当てることもできます。各ユーザーはプロジェクトまたはスペースに 1 つのロールしか持つことができませんが、プロジェクトとスペースごとに異なるロールを持つことができます。例えば、あるプロジェクトにはプロジェクト管理者ロールがあり、別のプロジェクトには寄稿者ロールがあります。

トピック

- スペースとプロジェクトのユーザーロールについて
- 各ロールで使用できるアクセス許可の表示
- ユーザーロールの表示と変更

スペースとプロジェクトのユーザーロールについて

スペースには 3 つのロールがあります。

- スペース管理者
- パワーユーザー
- 制限付きアクセス

プロジェクトへの招待を受け入れるユーザーには、プロジェクトを含むスペースで制限付きアクセスロールが自動的に割り当てられます。

プロジェクトのメンバーには、次の 4 つのロールを使用できます。

- プロジェクト管理者
- 寄稿者
- レビューワー
- 読み取り専用

プロジェクトにユーザーを追加すると、 CodeCatalyst によって自動的に制限付きアクセスロールが付与されます。すべてのプロジェクトからユーザーを削除すると、 CodeCatalyst はそのユーザーから制限付きアクセスロールを自動的に削除します。

スペース管理者ロール

Space 管理者ロールは、 で最も強力なロールです CodeCatalyst。スペース管理者ロールは、スペースのあらゆる側面を管理する必要があるユーザーにのみ割り当てます。このロールには、 のすべてのアクセス許可があるためです CodeCatalyst。Space 管理者ロールを持つユーザーは、Space 管理者ロールに他のユーザーを追加または削除したり、スペースを削除したりできる唯一のユーザーです。

スペースを作成すると、 によって CodeCatalyst 自動的にスペース管理者ロールが割り当てられます。ベストプラクティスとして、元のスペースクリエーターが利用できない場合に、このロールをこのロールで動作できる少なくとも 1 人の他のユーザーに追加することをお勧めします。

パワーユーザーロール

パワーユーザーロールはスペース内で CodeCatalyst 2 番目に強力なロールですが、スペース内のプロジェクトにはアクセスできません。これは、スペースにプロジェクトを作成し、スペースのユーザーとリソースを管理するのに役立つ必要があるユーザー向けに設計されています。作業の一環としてプロジェクトを作成し、スペース内のユーザーを管理する能力を必要とするチームリーダーまたはマネージャーであるユーザーに Power ユーザーロールを割り当てます。

制限付きアクセスロール

制限付きアクセスロールは、ほとんどのユーザーが CodeCatalyst スペースで持つロールです。これは、ユーザーがスペース内のプロジェクトへの招待を受け入れると自動的に割り当てられるロールです。これにより、そのプロジェクトを含むスペース内で作業するために必要な限定的なアクセス許可

が提供されます。スペースに直接招待するユーザーには、制限付きアクセスロールを割り当てます。ただし、作業でスペースの一部を管理する必要がある場合は除きます。

プロジェクト管理者ロール

プロジェクト管理者ロールは、プロジェクトで最も強力なロールです CodeCatalyst。このロールは、プロジェクト設定の編集、プロジェクトのアクセス許可の管理、プロジェクトの削除など、プロジェクトのあらゆる側面を管理する必要があるユーザーにのみ割り当てます。

プロジェクトロールには、スペースレベルでのアクセス許可はありません。したがって、プロジェクト管理者ロールを持つユーザーは、追加のプロジェクトを作成できません。Space 管理者または Power ユーザーロールを持つユーザーのみがプロジェクトを作成できます。

> **Note**
>
> Space 管理者ロールには、 のすべてのアクセス許可があります CodeCatalyst。

寄稿者ロール

Contributor ロールは、 CodeCatalyst プロジェクト内のメンバーの大部分を対象としています。このロールは、プロジェクト内のコード、ワークフロー、問題、アクションを操作できる必要があるユーザーに割り当てます。

レビューワーロール

Reviewer ロールは、プルリクエストや問題など、プロジェクト内のリソースとやり取りできるが、コードの作成とマージ、ワークフローの作成、 CodeCatalyst プロジェクト内のワークフロー実行の開始と停止はできないユーザーを対象としています。プルリクエストの承認とコメント、問題の作成、更新、解決、コメント、プロジェクト内のコードとワークフローの表示を行える必要があるユーザーにレビューワーロールを割り当てます。

読み取り専用ロール

読み取り専用ロールは、リソースとリソースのステータスを表示する必要があるが、リソースとやり取りしたり、プロジェクトに直接貢献したりしないユーザーを対象としています。このロールを持つユーザーは CodeCatalyst、 でリソースを作成することはできませんが、リポジトリのクローン作成やローカルコンピュータへの問題への添付ファイルのダウンロードなど、リソースを表示してコピーできます。リソースとプロジェクトの状態を表示する必要があるが、直接操作しないユーザーに読み取り専用ロールを割り当てます。

各ロールで使用できるアクセス許可の表示

次の表は、各 CodeCatalyst ロールで使用できるアクセス許可を示しています。リンクを使用して、適切なアクセス許可のセットにジャンプします。

- Space permissions
- Extensions permissions
- Project permissions
- Source repository permissions
- Dev Environment permissions
- Package repository and package permissions
- Workflow permissions
- Issues permissions
- Custom blueprint permissions
- Notifications permissions
- Search permissions

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
スペースのアクセス許可							
スペースの作成	✓	✗	✗	✗	✗	✗	✗
スペース請求の詳細を編集する	✓	✗	✗	✗	✗	✗	✗
シングルサインオンをセットアップ	✓	✗	✗	✗	✗	✗	✗

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
して有効にする							
シングルサインオンを削除する	✓	✗	✗	✗	✗	✗	✗
スペースの生成 AI 機能を有効にする	✓	✗	✗	✗	✗	✗	✗
スペースの生成 AI 機能を無効にする	✓	✗	✗	✗	✗	✗	✗
スペースの削除	✓	✗	✗	✗	✗	✗	✗
Space 管理者ロールに他のユーザーを追加する	✓	✗	✗	✗	✗	✗	✗
Space 管理者ロールから他のユーザーを削除する	✓	✗	✗	✗	✗	✗	✗

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
チームを作成する							
チームを削除する							
チームの更新							
スペースのマシンリソースを無効にする							
スペースのマシンリソースを有効にする							
プロジェクトの作成							
AWS アカウント接続をスペースに関連付ける							

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
AWS アカウント接続を更新する	✓	✓	✗	✗	✗	✗	✗
スペースから AWS アカウント接続の関連付けを解除する	✓	✓	✗	✗	✗	✗	✗
AWS アカウント接続を削除してスペースから削除する	✓	✓	✗	✗	✗	✗	✗
他のユーザーをスペースに招待する	✓	✓	✗	✗	✗	✗	✗
VPC 接続の作成	✓	✓	✗	✗	✗	✗	✗
VPC 接続を編集する	✓	✓	✗	✗	✗	✗	✗

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
VPC 接続を削除する	✓	✓	✗	✗	✗	✗	✗
スペース内のアクティビティのログを表示する	✓	✓	✗	✗	✗	✗	✗
AWS アカウント接続を表示する	✓	✓	✓	✓	✓	✓	✓
のインシデントを表示する CodeCatalyst	✓	✓	✓	✓	✓	✓	✓
表示スペース	✓	✓	✓	✓	✓	✓	✓
チームを表示する	✓	✓	✓	✓	✓	✓	✓
VPC 接続の表示	✓	✓	✓	✓	✓	✓	✓

拡張機能のアクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
拡張機能をインストールする							
拡張機能の更新							
拡張機能を削除する							
GitHub アカウントを接続する							
GitHub アカウントの切断							
Jira サイトを接続する							
Jira サイトを切断する							

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
インストールされた拡張機能の設定の詳細を表示する							
拡張機能の表示							

プロジェクトのアクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
プロジェクト設定の編集							
プロジェクトのマシンリソースを無効にする							
プロジェクトのマシンリソースを有効にする							

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
プロジェクトの削除							
プロジェクトにユーザーを招待する							
プロジェクトのユーザーのロールを変更する							
プロジェクトからユーザーを削除する							
プロジェクトにチームを追加する							
プロジェクトからチームを削除する							

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
チームのプロジェクトロールを変更する							
プロジェクトを表示する							
プロジェクトアクティビティの表示							
プロジェクト内のチームを表示する							
ブループリントを表示する							

ソースリポジトリのアクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
リポジトリの作成							

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
リンクリポジトリ							
リポジトリのリンク解除							
リポジトリを削除する							
リポジトリ設定の編集							
リポジトリの表示							
リポジトリ設定の表示							
リポジトリのクローンを作成する							
ブランチの作成							
ブランチルールを作成する							

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
デフォルトのブランチを変更する							
ブランチを削除する							
ブランチをマージ							
ブランチルールを更新する							
ブランチの表示							
ブランチルールの表示							
フォルダの作成							
フォルダの削除							
フォルダの編集							
フォルダの表示							

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
ファイルの作成							
ファイルを削除							
ファイルの編集							
ファイルの表示							
コミットの作成とプッシュ							
コミットの表示							
プルリクエストを作成する							
プルリクエストの承認ルールを作成する							
プルリクエストのマージ要件を上書きする							

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
プルリクエストを更新する							
プルリクエストの承認ルールを更新する							
プルリクエストの表示							
プルリクエストの承認ルールを表示する							
プルリクエストを閉じる							
プルリクエストを承認する							
プルリクエストに関するコメント							

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
プルリクエストに関するコメントでAmazon Q を操作する							
Amazon Q によって作成されたプルリクエストのリビジョンを作成する							
プルリクエストに問題のリンク							
プルリクエストから問題のリンクを解除する							

開発環境のアクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
独自の開発環境を作成する							
独自の開発環境を停止する							
他のユーザーが作成した開発環境を停止する							
独自の開発環境を再開する							
独自の開発環境を表示する							
他のユーザーによって作成された開発環境を表示する							
独自の開発環境を編集する							

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
他のユーザーが作成した開発環境を編集する							
独自の開発環境を削除する							
他のユーザーが作成した開発環境を削除する							
開発環境用の devfile を作成する							
開発環境の devfile を編集する							
開発環境用の devfile を削除する							

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
開発環境用の devfile を表示する							

パッケージリポジトリとパッケージのアクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
パッケージリポジトリを作成する							
パッケージリポジトリの表示							
パッケージリポジトリの編集							
パッケージリポジトリを削除する							

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
ゲートウェイパッケージリポジトリを作成する							
ゲートウェイパッケージリポジトリの表示							
ゲートウェイパッケージリポジトリを削除する							
アップストリームパッケージリポジトリの追加							

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
アップストリームリポジトリの検索順序を編集する							
アップストリームパッケージリポジトリを削除する							
パッケージリポジトリに接続する							
パッケージリポジトリからパッケージを読み取る							
パッケージリポジトリへのパッケージの発行							

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
アップストリームリポジトリからパッケージを読み取って保持する							
パッケージの表示							
パッケージバージョンを表示する							
パッケージバージョンのアセットを表示する							
パッケージバージョンの依存関係を一覧表示する							

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
パッケージバージョンのステータスの更新							
パッケージオリジン設定を更新する							
パッケージバージョンを削除する							

ワークフローのアクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
ワークフローの作成							
ワークフローの更新							
ワークフローを削除する							

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
ワークフローを開始する							
ワークフローを停止する							
ワークフローシークレットを作成する							
ワークフローシークレットを更新する							
ワークフローシークレットを削除する							
環境の作成							
環境を削除する							
フリートの作成							

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
フリートを更新する	✓	✗	✗	✓	✗	✗	✗
フリートを削除する	✓	✗	✗	✓	✗	✗	✗
他のアカウントのワークフローリソースを管理する	✓	✗	✗	✓	✗	✗	✗
VPC 接続を環境に関連付ける	✓	✗	✗	✓	✗	✗	✗
VPC 接続と環境の関連付けを解除する	✓	✗	✗	✓	✗	✗	✗
VPC に接続された環境をワークフローに関連付ける	✓	✗	✗	✓	✓	✗	✗

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
VPC 接続環境とワークフローの関連付けを解除する							
ワークフロー実行を承認する							
ワークフロー内のコミットを追跡する							
環境を表示する							
ビルドアクションログを表示する							
フリートを表示する							
テストアクションログの表示							

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
ワークフローの表示							
ワークフロー実行の表示							
ワークフロー実行結果の表示							
ワークフローシークレットを表示する							

アクセス許可を発行する	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
問題の作成							
更新の問題							
問題を表示する							
問題をアーカイブする							

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
Amazon Q に問題を割り当てる							
問題に関するコメントで Amazon Q とやり取りする							
問題から Amazon Q の割り当てを解除する							
他のユーザーによって作成された問題を更新する							
問題に関するコメントを表示する							
問題に関するコメントを作成する							

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
問題に関するコメントを更新する							
ラベルの作成							
ラベルを更新する							
ラベルの表示							
問題にラベルを追加する							
問題からラベルを削除する							
問題のカスタムステータスを作成する							
カスタムステータスを更新する							

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
カスタムステータスを表示する							
カスタムステータスの移動							
カスタムステータスを非アクティブ化する							
問題に添付ファイルを追加する							
問題の添付ファイルを表示する							
問題から添付ファイルを削除する							
プルリクエストを問題にリンクする							

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
プルリクエストを問題からリンク解除する	✓	✗	✗	✓	✓	✗	✗
Jira プロジェクトをリンクする	✓	✗	✗	✓	✗	✗	✗
Jira プロジェクトのリンクを解除する	✓	✗	✗	✓	✗	✗	✗

カスタムブループリントのアクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
カスタムブループリントプロジェクトを作成する	✓	✓	✗	✗	✗	✗	✗

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
プレビューカスタムブループリントを公開する							
プレビューカスタムブループリントの公開解除							
カスタムブループリントを公開する							
カスタムブループリントの公開解除							
スペースブループリントカタログにカスタムブループリントを追加する							

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
スペースブループリントカタログからカスタムブループリントを削除する	✓	✓	✗	✗	✗	✗	✗
カスタムブループリントの公開アクセス許可を管理する	✓	✓	✗	✗	✗	✗	✗
カスタムブループリントのカタログバージョンを管理する	✓	✓	✗	✗	✗	✗	✗
カスタムブループリントを更新する	✓	✓	✗	✗	✗	✗	✗

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
カスタムブループリントバージョンを削除する	✓	✓	✗	✗	✗	✗	✗
カスタムブループリントを削除する	✓	✓	✗	✗	✗	✗	✗
カスタムブループリントをプロジェクトに適用する	✓	✓	✗	✓	✗	✗	✗
カスタムブループリントとプロジェクトの関連付けを解除する	✓	✓	✗	✓	✗	✗	✗

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
適用されたカスタムブループリントのバージョンを更新する	✓	✓	✗	✓	✗	✗	✗
カスタムブループリントのエイリアスを編集する	✓	✓	✗	✓	✗	✗	✗
公開されたカスタムブループリントを表示する	✓	✓	✓	✓	✓	✓	✓

通知のアクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
通知チャネルを設定する	✓	✗	✗	✗	✗	✗	✗
通知チャネルを削除する	✓	✗	✗	✗	✗	✗	✗

アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
通知設定の編集							
通知設定の表示							
CodeCatalyst インシデントに関する通知を自動的に受信する							
関連付けられた E メールアカウントの E メール通知を設定する							

検索アクセス許可	スペース管理者ロール	パワーユーザーロール	制限付きアクセスロール	プロジェクト管理者ロール	寄稿者ロール	レビューワーロール	読み取り専用ロール
プロジェクト内で検索する							
スペース全体を検索する							

ユーザーロールの表示と変更

ユーザーに割り当てられたロールを表示できます。これにより、プロジェクトで実行できるアクションを理解できます。追加のアクセス許可が必要な場合は、ロールを変更することもできます。

プロジェクト内のユーザーのロールを表示するには

1. 各プロジェクトメンバーに関連付けられたロールを表示するプロジェクトに移動します。

   > **Tip**
   > 上部のナビゲーションバーに表示するプロジェクトを選択できます。

2. ナビゲーションペインで、プロジェクト設定 を選択します。
3. メンバータブでは、各プロジェクトメンバーのロールがロール に表示されます。

プロジェクトでユーザーのロールを変更するには

1. プロジェクトメンバーに関連付けられたロールを変更するプロジェクトに移動します。

   > **Tip**
   > 上部のナビゲーションバーに表示するプロジェクトを選択できます。

2. ナビゲーションペインで、プロジェクト設定 を選択します。
3. メンバータブのプロジェクトメンバー で、ロールを変更するユーザーを選択します。アクション を選択し、ロールの編集 を選択します。
4. ロール でプロジェクトロールを選択し、確認 を選択します。

スペースでのロールの表示と変更

でプロジェクトへの招待を受け入れるすべてのユーザーは、プロジェクトのスペースのメンバー CodeCatalyst になります。スペースメンバーのリストを表示できます。スペース管理者への制限付きアクセスからユーザーロールを変更して、スペースとそのリソースをより適切に管理できます。Space 管理者ロールは、ユーザーが でプロジェクトを作成できるようにする唯一のロールです CodeCatalyst。

> **Warning**
>
> Space 管理者ロールは、 で最も強力なロールです CodeCatalyst。このロールを持つユーザーは、スペースの削除など CodeCatalyst、 で任意のアクションを実行できます。このロールは、スペースへのこのレベルのアクセスを必要とするユーザーにのみ割り当てます。詳細については、「スペース管理者ロール」を参照してください。

スペース内のユーザーロールを変更するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. スペースに移動します。

   > **Tip**
   >
   > 複数のスペースに所属している場合は、表示するスペースを上部のナビゲーションバーで選択できます。

3. [メンバー] タブを選択します。
4. ロールを変更するユーザーを選択し、ロールの変更 を選択します。
5. ロールの変更 で、割り当てるロールを選択し、確認 を選択します。

個人用アクセストークンを使用してリポジトリアクセスをユーザーに付与する

Git クライアントまたは統合開発環境 (IDE) を備えたローカルコンピュータで、ソースリポジトリなどの一部の CodeCatalystリソースにアクセスするには、アプリケーション固有のパスワードを入力する必要があります。この目的で使用する個人アクセストークン (PAT) を作成できます。作成した PATsは、 のすべてのスペースとプロジェクトにわたってユーザー ID に関連付けられます CodeCatalyst。 CodeCatalyst ID には複数の PAT を作成できます。

作成した PATs の名前と有効期限を表示したり、不要になった PAT を削除したりできます。PAT シークレットは、作成時にのみコピーできます。

> **Note**
>
> デフォルトでは、PATs 1 年で期限切れになります。

PATsの作成

PATsは、 のユーザー ID に関連付けられます CodeCatalyst。PAT シークレットは、作成時にのみコピーできます。

PATsの作成 (コンソール)

コンソールを使用して、 で PATs を作成できます CodeCatalyst。

個人用アクセストークンを作成するには (コンソール)

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 上部のメニューバーでプロファイルバッジを選択し、[My 設定] を選択します。 CodeCatalyst マイ設定ページが開きます。

   > **Tip**
   >
   > ユーザープロファイルは、プロジェクトまたはスペースのメンバーページに移動し、メンバーリストから自分の名前を選択することで検索することもできます。

3. [個人アクセストークン] で [作成] を選択します。

   PAT の作成ページが表示されます。
4. PAT 名 に、PAT のわかりやすい名前を入力します。
5. [有効期限] では、デフォルトの日付のままにしておくか、カレンダーアイコンを選択して、カスタムの日付を選択します。有効期限のデフォルトは、現在の日付から 1 年です。
6. [作成] を選択します。

   > **Tip**
   >
   > このトークンは、ソースリポジトリの [クローンリポジトリC] を選択したときにも作成できます。

7. PAT シークレットをコピーするには、コピー を選択します。取得できる PAT シークレットを保存します。

> **Important**
>
> PAT シークレットは 1 回だけ表示されます。ウィンドウを閉じた後に取得することはできません。PAT シークレットを安全な場所に保存しなかった場合は、別のシークレットを作成できます。

.

PATsの作成 (CLI)

CLI を使用して、 で PATs を作成できます CodeCatalyst。

個人用アクセストークンを作成するには (AWS CLI)

1. ターミナルまたはコマンドラインで、次のようにcreate-access-tokenコマンドを実行します。

```
aws codecatalyst create-access-token
```

成功すると、コマンドは次の例のように、作成された PAT に関する情報を返します。

```
{
    "secret": "value",
    "name": "marymajor-22222EXAMPLE",
    "expiresTime": "2024-02-04T01:56:04.402000+00:00"
}
```

2.

PAT シークレットは、PAT の作成時に 1 回しか表示できません。PAT シークレットを紛失した場合、または安全に保存されていないことが懸念される場合は、別のシークレットを作成できます。

を使用して、ユーザーアカウントに関連付けられた PATs を表示できます AWS CLI。PAT に関する情報のみを表示でき、PAT シークレット自体の値を表示することはできません。

> **Note**
>
> AWS CLI を使用するには、最新バージョンの を使用していることを確認してください CodeCatalyst。以前のバージョンには CodeCatalyst コマンドが含まれていない場合があります。 AWS CLI プロファイルは、 で使用する前に設定する必要があります CodeCatalyst。詳細については、「AWS CLIとを使用するためのセットアップ CodeCatalyst」を参照してください。

PATsの表示

PATs を表示できます CodeCatalyst。このリストには、ユーザー ID に関連付けたすべての PATsが表示されます。PAT は、 のすべてのスペースとプロジェクトにわたってユーザープロファイルに関連付けられます CodeCatalyst。期限切れの PATs、期限切れ後に削除されるため表示されません。

PATs の表示 (コンソール)

コンソールを使用して、 でユーザー ID に関連付けられた PATs を表示できます CodeCatalyst。

個人用アクセストークンを表示するには (コンソール)

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 上部のメニューバーでプロファイルバッジを選択し、[My 設定] を選択します。 CodeCatalyst マイ設定ページが開きます。

   > **Tip**
   >
   > ユーザープロファイルは、プロジェクトまたはスペースのメンバーページに移動し、メンバーリストから自分の名前を選択することで検索することもできます。

3. 「個人用アクセストークン」で、現在の PATsの名前と有効期限を表示します。

PATs の表示 (CLI)

CLI を使用して、 でユーザー ID に関連付けられた PATs を表示できます CodeCatalyst。

個人用アクセストークンを表示するには (AWS CLI)

- ターミナルまたはコマンドラインで、次のようにlist-access-tokensコマンドを実行します。

```
aws codecatalyst list-access-tokens
```

成功すると、コマンドは次の例のようにユーザーアカウントに関連付けられた PATs に関する情報を返します。

```
{
    "items": [
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
            "name": "marymajor-22222EXAMPLE",
            "expiresTime": "2024-02-04T01:56:04.402000+00:00"
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
            "name": "marymajor-11111EXAMPLE",
            "expiresTime": "2023-03-12T01:58:40.694000+00:00"
        }
    ]
}
```

PATsの削除

でユーザー ID に関連付けられた PATs を削除できます CodeCatalyst。

PATsの削除 (コンソール)

コンソールを使用して、 で PATs を削除できます CodeCatalyst。

個人用アクセストークンを削除するには (コンソール)

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 上部のメニューバーでプロファイルバッジを選択し、[My 設定] を選択します。 CodeCatalyst マイ設定ページが開きます。

> **Tip**
>
> ユーザープロファイルは、プロジェクトまたはスペースのメンバーページに移動し、メンバーリストから自分の名前を選択することで検索することもできます。

3. 「個人用アクセストークン」で、削除する PAT の横にあるセレクターを選択し、「削除」を選択します。

   PAT の削除: <name>? ページで、削除を確認するには、テキストフィールドに delete と入力します。[削除] を選択します。

PATsの削除 (CLI)

を使用して、ユーザー ID に関連付けられた PAT を削除できます AWS CLI。これを行うには、PAT の ID を指定する必要があります。この ID は、 delete-access-token コマンドを使用して表示できます。

> **Note**
>
> で作業 AWS CLI するには、最新バージョンの を使用していることを確認してください CodeCatalyst。以前のバージョンには CodeCatalyst コマンドが含まれていない場合があります。 AWS CLI で を使用する方法の詳細については、 CodeCatalyst「」を参照してくださいAWS CLIとを使用するためのセットアップ CodeCatalyst。

個人用アクセストークンを削除するには (AWS CLI)

- ターミナルまたはコマンドラインで、 delete-access-token コマンドを実行し、削除する PAT の ID を指定します。例えば、次のコマンドを実行して、ID が *123EXAMPLE* の PAT を削除します。

  ```
  aws codecatalyst delete-access-token --id a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb
  ```

  成功した場合、このコマンドはレスポンスを返しません。

個人接続による GitHub リソースへのアクセス

個人用接続を使用して、サードパーティー GitHub リソースを認証し、 に接続できます CodeCatalyst。例えば、個人用接続を使用して、 CodeCatalyst が アカウント GitHubにアクセスし、プロジェクトまたはブループリントのソースとしてリポジトリを作成することを に許可します。接続は CodeCatalyst ID にマッピングされ、1 つ以上のソースリポジトリに接続するために使用

できます。作成した接続は、 のすべてのスペースとプロジェクトにわたってユーザー ID に関連付けられます CodeCatalyst。

> **Note**
>
> ブループリントを使用して、アクセスできる GitHub 組織の個人接続を管理できます。

プロバイダータイプごとに、すべてのスペースで 1 つのユーザー ID (CodeCatalyst エイリアス) に対して 1 つの個人用接続を作成できます。

の個人用接続を使用して CodeCatalyst 、プロジェクトの GitHub リポジトリを作成し、ブループリントの GitHub ソースリポジトリを選択し、 GitHub リポジトリ CodeCatalyst のプルリクエストを管理できます。

> **Note**
>
> 設計図を GitHub リポジトリに関連付けるための個人用接続の使用は、 GitHub リポジトリをリンク CodeCatalyst するための での拡張機能の使用とは異なります。拡張機能の詳細については、「」を参照してください[で拡張機能を使用してプロジェクトに機能を追加する CodeCatalyst]。

個人用接続の作成

コンソールを使用して、 でユーザー ID に関連付けられた個人用接続を作成できます CodeCatalyst。

個人用接続を作成するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 上部のメニューバーでプロファイルバッジを選択し、[My 設定] を選択します。 CodeCatalyst マイ設定ページが開きます。

> **Tip**
>
> ユーザープロファイルは、プロジェクトまたはスペースのメンバーページに移動し、メンバーリストから自分の名前を選択することで検索することもできます。

3. 「個人用接続」で、「 の作成」を選択します。

   接続の作成ページが表示されます。

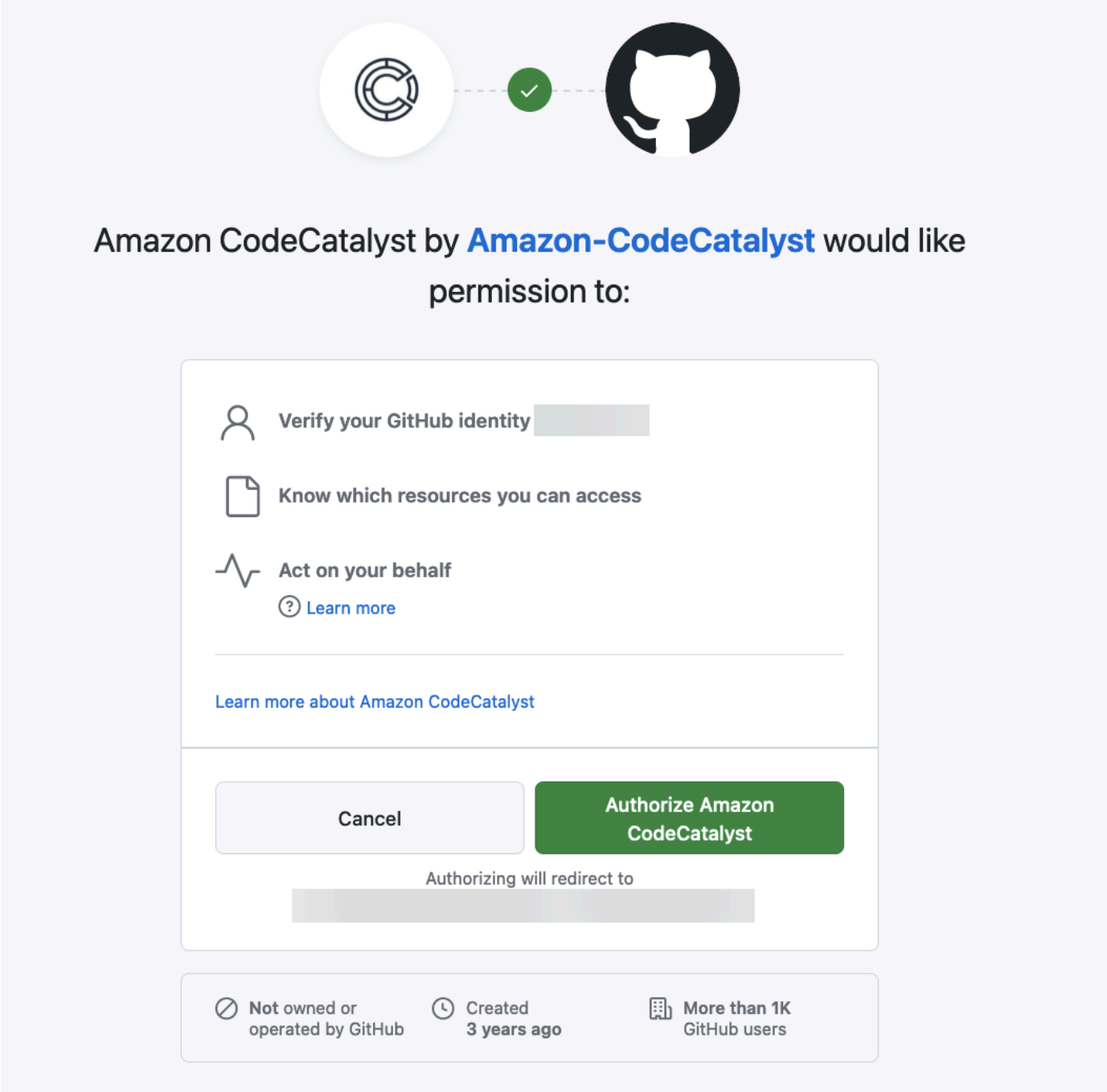
4. [作成] を選択します。接続の作成ページが表示されます。

5. 「接続の作成」ページの「プロバイダー」で、「」を選択しますGitHub。接続名 で、接続の名前を入力します。[作成] を選択します。

6. プロンプトが表示されたら、 GitHub アカウントにサインインします。

7. 接続確認ページで、「受け入れ」を選択します。

8. インストール確認ページで認証ボタンを選択して、コネクタアプリケーションをインストールすることを確認します。



個人用接続の削除

でユーザー ID に関連付けられた個人接続を削除できます CodeCatalyst。

> **Note**
>
> で個人接続を削除 CodeCatalyst しても、 GitHub アカウント内のアプリケーションはアンインストールされません。新しい個人用接続を作成する場合は、アプリのインストールを使用できます。でアプリケーションをアンインストールするには GitHub、アプリケーションを取り消し、後で再インストールします。

で個人接続を削除するには CodeCatalyst

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 上部のメニューバーでプロファイルバッジを選択し、[My 設定] を選択します。 CodeCatalyst マイ設定ページが開きます。

   > **Tip**
   >
   > ユーザープロファイルは、プロジェクトまたはスペースのメンバーページに移動し、メンバーリストから自分の名前を選択することで検索することもできます。

3. 「個人用接続」で、削除する接続の横にあるセレクターを選択し、「削除」を選択します。

   「接続の削除: <name>?」ページで、削除を確認するには、テキストフィールドに「削除」と入力します。[削除] を選択します。

1. にサインイン GitHub し、インストールされているアプリのアカウント設定に移動します。プロファイルアイコンを選択し、設定 を選択し、アプリケーション を選択します。
2. 承認された GitHub アプリケーション タブの承認されたアプリケーションのリストで、 にインストールされているアプリケーションを表示します CodeCatalyst。インストールを取り消すには、「 を取り消す」を選択します。

多要素認証 (MFA) でサインインするように AWS Builder ID を設定する

Builder ID AWS プロファイルを個人使用用でもプロフェッショナル使用用でも、多要素認証 (MFA) を別のセキュリティレイヤーとして設定することをお勧めします。スペースのメンバーであり、プ

ロジェクトで他のユーザーとコラボレーションする場合は、特に MFA を設定することをお勧めします。プロジェクトには複数のユーザーがアクセスできるため、セキュリティ侵害の機会が増えます。

MFA を有効にするときは、E メールとパスワード CodeCatalyst を使用して Amazon にサインインする必要があります。サインインのこの部分は、既知のものを使用する最初の要素です。次に、コードまたはセキュリティキーを使用してサインインします。これは 2 番目の要素であり、ユーザーが持っているものです。2 つ目の要素は、モバイルデバイスによって生成される認証コード、またはコンピュータに接続されたセキュリティキーをタップまたは押すことによって生成される認証コードです。これら複数の要因を組み合わせることで、不正アクセスを防止することでセキュリティを強化できます。

多要素認証用のデバイスを登録する方法

マイプロファイル > 多要素認証で次の手順を使用して、新しいデバイスを多要素認証 (MFA) に登録します。

> **Note**
>
> この手順のステップを開始する前に、まず適切な認証アプリをデバイスにダウンロードすることをお勧めします。MFA デバイスに使用できるアプリケーションの一覧については、「認証アプリケーション」を参照してください。

MFA を使用するデバイスを登録するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 右上で、最初の頭文字が付いたアイコンの横にある矢印を選択し、ユーザープロファイル を選択します。 CodeCatalyst プロファイルページが開きます。
3. プロファイルページで、プロファイルとセキュリティの管理を選択します。AWS Builder ID プロファイルページが開きます。
4. ページの左側で、セキュリティ を選択します。
5. 多要素認証ページで、デバイス の登録 を選択します。
6. MFA デバイスの登録ページで、次のいずれかの MFA デバイスタイプを選択し、指示に従ってください。

   - セキュリティキーと内蔵認証システム
     1. [Register your user's security key] (ユーザーのセキュリティキーの登録) ページでは、お使いのブラウザやプラットフォームの指示に従ってください。

> **Note**
>
> このエクスペリエンスはオペレーティングシステムとブラウザによって異なるため、ブラウザまたはプラットフォームに表示される指示に従ってください。デバイスが正常に登録されると、登録したデバイスに識別しやすい表示名を付けるオプションが表示されます。変更する場合は、[Rename] (名前の変更) を選択し、新しい名前を入力してから [Save] (保存) を選択します。

- 認証システムアプリケーション

  1. [Set up the authenticator app] (認証システムアプリケーションの設定) ページで、QR コードのグラフィックを含む新しい MFA デバイスの設定情報を表示します。図は、QR コードに対応していないデバイスのマニュアル入力に利用できるシークレットキーを示しています。
  2. 物理的に MFA デバイスを使用して、次の操作を行います。

     a. 互換性のある MFA 認証システムアプリケーションを開きます。MFA デバイスで使用できるテスト済みアプリケーションのリストについては、「テスト済みの認証アプリ」を参照してください。MFA アプリが複数のデバイスをサポートしている場合は、新しい MFA デバイスを作成するオプションを選択します。

     b. MFA アプリケーションが QR コードをサポートしているかどうかを判断し、[Set up the authenticator app] (認証アプリケーションの設定) ページで以下のいずれかの操作を行います。

        i. [Show QR code] (QR コードの表示) を選択し、アプリケーションを使用して QR コードをスキャンします。例えば、カメラアイコンまたは スキャンコード に似たオプションを選択します。次に、デバイスのカメラでコードをスキャンします。

        ii. [show secret key] (シークレットキーを表示する) をクリックし、そのシークレットキーを MFA アプリケーションに入力します。

> **Important**
>
> AWS Builder ID の MFA デバイスを設定するときは、QR コードまたはシークレットキーのコピーを安全な場所に保存します。これは、携帯電話を紛失した場合や、MFA 認証システムアプリケーションを再インストールしなければ

ならない場合に役立ちます。いずれの場合も、すぐにアプリケーションを再設定して同じ MFA 設定を使用することができます。

3. [Set up the authenticator app] (認証システムアプリケーションをセットアップする) ページで、[Authenticator code] (認証コード) で、物理的な MFA デバイスに現在表示されているワンタイムパスワードを入力します。

> **Important**
>
> コードを生成したら、即時にリクエストを送信します。コードを生成してからリクエストを送信するまで時間がかかりすぎる場合、MFA デバイスは Builder ID AWS プロファイルに正常に関連付けられますが、MFA デバイスは同期しません。これは、タイムベースドワンタイムパスワード (TOTP) の有効期間が短いために起こります。その場合は、デバイスの再同期ができます。

4. [Assign MFA] (MFA の割り当て) を選択します。MFA デバイスはワンタイムパスワードの生成を開始でき、使用できる状態になりました。

認証アプリケーション

認証アプリケーションは、ワンタイムパスワード (OTP) ベースのサードパーティー認証アプリケーションです。ユーザーは、モバイルデバイスやタブレットにインストールされた認証アプリケーションを、許可された MFA デバイスとして使用することができます。サードパーティー認証アプリケーションは、6 桁の認証コードを生成できる標準ベースの TOTP (タイムベースワンタイムパスワード) アルゴリズムである RFC 6238 に準拠している必要があります。

MFA を求めるプロンプトが表示されたら、ユーザーは認証アプリケーションから有効なコードを入力ボックスに入力する必要があります。ユーザーに割り当てられた各 MFA デバイスは一意であることが必要です。1 人の ユーザーに対して 2 つの認証アプリを登録することができます。

テスト済みの認証アプリ

TOTP 準拠のアプリケーションは IAM Identity Center MFA で動作しますが、次の表は、よく知られているサードパーティー認証アプリケーションから選択できます。

オペレーティングシステム	テスト済みの認証アプリ
Android	Authy、Duo Mobile、LastPass Authenticator、Microsoft Authenticator、Google Authenticator
iOS	Authy、Duo Mobile、LastPass Authenticator、Microsoft Authenticator、Google Authenticator

MFA デバイスの変更

MFA デバイスを登録したら、その名前を変更または削除できます。セキュリティを強化するために、少なくとも 1 つの MFA デバイスを常に有効にすることをお勧めします。最大 5 つのデバイスを登録できます。追加方法については、「」を参照してください多要素認証用のデバイスを登録する方法。

MFA デバイスの名前変更

MFA デバイスの名前を変更するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 右上で、最初の頭文字が付いたアイコンの横にある矢印を選択し、ユーザープロファイル を選択します。 CodeCatalyst プロファイルページが開きます。
3. プロファイルページで、プロファイルとセキュリティの管理を選択します。AWS Builder ID プロファイルページが開きます。
4. ページの左側にある多要素認証を選択します。ページに到着すると、名前の変更がグレー表示されます。
5. 変更する MFA デバイスを選択します。[名前の変更] を選択します。その後、モーダルがポップアップします。
6. 開いたプロンプトで、MFA デバイス名 に新しい名前を入力し、名前の変更 を選択します。名前が変更されたデバイスは、多要素認証デバイス (MFA) の下に表示されます。

MFA デバイスの削除

MFA デバイスを削除するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. 右上で、最初の頭文字が付いたアイコンの横にある矢印を選択し、ユーザープロファイル を選択します。 CodeCatalyst プロファイルページが開きます。
3. プロファイルページで、プロファイルとセキュリティの管理を選択します。AWS Builder ID プロファイルページが開きます。
4. ページの左側にある多要素認証を選択します。ページに到着すると、Delete がグレー表示されます。
5. 変更する MFA デバイスを選択します。[削除] を選択します。MFA デバイスを削除するというモーダルが表示されます。手順に従ってデバイスを削除します。
6. [削除] を選択します。削除されたデバイスは、多要素認証デバイス (MFA) に表示されなくなります。

Amazon のセキュリティ CodeCatalyst

のクラウドセキュリティが最優先事項 AWS です。お客様は AWS 、セキュリティを最も重視するスペースの要件を満たすように構築されたデータセンターとネットワークアーキテクチャからメリットを得られます。

セキュリティは、 AWS とユーザーの間で共有される責任です。責任共有モデルでは、これをクラウドのセキュリティおよびクラウド内のセキュリティと説明しています。

- クラウドのセキュリティ — AWS は、 で AWS サービスを実行するインフラストラクチャを保護する責任を担います AWS クラウド。また、 は、お客様が安全に使用できるサービス AWS も提供します。コンプライアンスAWS プログラムコンプライアンスプログラム の一環として、サードパーティーの監査者は定期的にセキュリティの有効性をテストおよび検証。に適用されるコンプライアンスプログラムの詳細については CodeCatalyst、「コンプライアンスプログラムAWS による対象範囲内のサービスコンプライアンスプログラム」を参照してください。
- クラウド内のセキュリティ — お客様の責任は、使用する AWS サービスによって決まります。また、お客様は、データの機密性、会社の要件、適用される法律や規制など、その他の要因についても責任を負います。

このドキュメントは、Amazon の使用時に責任共有モデルを適用する方法を理解するのに役立ちます CodeCatalyst。セキュリティおよびコンプライアンスの目的 CodeCatalyst を達成するために を設定する方法を示します。また、 リソースのモニタリングや保護に役立つ他の AWS のサービスの使用方法についても CodeCatalyst 説明します。

内容

Amazon データ保護 CodeCatalyst

CodeCatalyst セキュリティとコンプライアンスはAmazonと顧客間の責任分担です。AWShttps://aws.amazon.com/compliance/shared-responsibility-model/、。このモデルで説明したように、CodeCatalyst はサービスのグローバルインフラストラクチャを保護する責任があります。顧客は、このインフラストラクチャでホストされているコンテンツに対する管理を維持する責任があります。この責任分担モデルは、のデータ保護にも適用されます CodeCatalyst。

データ保護のため、アカウントの認証情報を保護し、サインイン時に多要素認証を設定することをお勧めします。詳細については、「多要素認証 (MFA) でサインインするように AWS Builder ID を設定する」を参照してください。

タグや名前フィールドなどの自由形式のフィールドには、顧客のメールアドレスなどの機密情報や機密情報を入力しないでください。これには、リソース名やユーザーが入力するその他の識別子のほか、接続されている識別子も含まれます。 CodeCatalyst AWS アカウントたとえば、スペース、プロジェクト、またはデプロイフリート名の一部に機密情報や機密情報を入力しないでください。タグ、名前、または名前に使用される自由形式のフィールドに入力したデータは、請求ログや診断ログに使用されたり、URL パスに含まれたりする可能性があります。これは、コンソール、API、CodeCatalyst アクション開発キットAWS CLI、AWSまたは任意の SDK を使用する場合に当てはまります。

外部サーバーに URL を提供する場合、そのサーバーへのリクエストを検証するためのセキュリティ認証情報を URL に含めないことを強くお勧めします。

CodeCatalyst ソースリポジトリは保存時に自動的に暗号化されます。お客様によるアクションは不要です。 CodeCatalyst また、HTTPS プロトコルを使用して転送中のリポジトリデータを暗号化します。

CodeCatalyst MFA をサポートします。詳細については、「多要素認証 (MFA) でサインインするように AWS Builder ID を設定する」を参照してください。

データ暗号化

CodeCatalyst データをサービス内に安全に保存して転送します。すべてのデータは、転送時と保管時のいずれも暗号化されます。サービスのメタデータを含め、サービスによって作成または保存されるデータはすべて、サービスにネイティブに保存され、暗号化されます。

> **Note**
>
> 課題に関する情報はサービス内に安全に保存されますが、未解決の課題に関する情報は、課題掲示板、バックログ、個別の課題を表示したブラウザーのローカルキャッシュにも保存されます。セキュリティを最大限に高めるには、ブラウザのキャッシュをクリアしてこの情報を削除してください。

AWS アカウントへのアカウント接続や内のリンクされたリポジトリなど CodeCatalyst、リンクされたリソースを使用する場合 GitHub、 CodeCatalyst リンク先のリソースから転送されるデータは暗号化されますが、リンクされたリソースのデータ処理はそのリンクされたサービスによって管理されます。詳細については、リンク先サービスのドキュメントとを参照してくださいAmazon のワークフローアクションのベストプラクティス CodeCatalyst。

キーの管理

CodeCatalyst キー管理はサポートしていません。

ネットワーク間トラフィックのプライバシー

でスペースを作成するときに CodeCatalyst、AWS リージョンそのスペースのデータとリソースを保存する場所を選択します。プロジェクトデータとメタデータがそこから出ることはありませんAWS リージョン。ただし、内部のナビゲーションをサポートするために CodeCatalyst、限られたスペー

ス、プロジェクト、AWS リージョンおよびユーザーメタデータがパーティション内のすべてに複製されます。AWS リージョンそのパーティションの外部には複製されません。たとえば、AWS リージョンスペースの作成時に米国西部 (オレゴン) を選択した場合、データは中国地域のリージョンまたはには複製されません。AWS GovCloud (US)詳細については、「管理、AWSグローバルインフラストラクチャAWS リージョン、AWSおよびサービスエンドポイント」を参照してください。

AWS リージョンパーティション内で複製されるデータには以下が含まれます。

- スペース名が一意であることを保証するために、スペースの名前を表す暗号化されたハッシュ値。この値は人間が読めるものではなく、実際のスペースの名前も公開されません。
- スペースのユニーク ID
- スペース間のナビゲーションに役立つスペースのメタデータ。
- AWS リージョンスペースが配置されている場所
- スペース内のすべてのプロジェクトの固有 ID
- スペースまたはプロジェクトにおけるユーザーの役割を示すロール ID
- サインアップ時に CodeCatalyst、サインアッププロセスに関するデータとメタデータ。これには以下が含まれます。
  - のユニーク ID AWS ビルダー ID
  - 内のユーザーの表示名。AWS ビルダー ID
  - そのユーザーのエイリアス。AWS ビルダー ID
  - ユーザーがサインアップしたときに使用したメールアドレス AWS ビルダー ID
  - サインアッププロセスの進行状況。
  - サインアッププロセスの一環としてスペースを作成する場合、そのスペースの請求アカウントとして使用される AWS アカウント ID

CodeCatalystスペース名は全体で一意です。スペース名には機密データを含めないようにしてください。

リンクされたリソースや接続されたアカウント (AWS アカウント GitHub またはリポジトリへの接続など) を扱う場合は、ソースとターゲットのロケーションをそれぞれがサポートする最高レベルのセキュリティで設定することをおすすめします。 CodeCatalyst トランスポート層セキュリティ (TLS) 1.2 を使用してAWS アカウントAWS リージョン、、とアベイラビリティーゾーン間の接続を保護します。

Identity and Access Management と Amazon CodeCatalyst

Amazon では CodeCatalyst、サインインしてスペースとプロジェクトにアクセスするために Builder ID AWS を作成して使用します。 AWS Builder ID は AWS Identity and Access Management (IAM) のアイデンティティではなく、 には存在しません AWS アカウント。ただし、請求目的でスペースを検証する場合や、 に接続してその でリソースを作成および使用する場合 AWS アカウント 、CodeCatalyst は IAM と統合されます AWS アカウント。

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御 AWS のサービス するのに役立つ です。IAM 管理者は、リソースを使用するための認証 (サインイン) および許可 (アクセス許可を持たせる) を行うことができる人を制御します。IAM は、追加料金なしで AWS のサービス 使用できる です。

Amazon でスペースを作成するときは CodeCatalyst、スペースの請求アカウントとして を接続する AWS アカウント 必要があります。 CodeCatalyst スペースを検証するには AWS アカウント 、 に管理者権限を持っているか、 権限を持っている必要があります。また、接続された でリソースを作成およびアクセスするために CodeCatalyst が使用できるスペースに IAM ロールを追加することもできます AWS アカウント。これはサービスロール と呼ばれます。複数の への接続を作成し AWS アカウント 、それらの各アカウント CodeCatalyst で のサービスロールを作成することもできます。

> **Note**
>
> の請求 CodeCatalyst は、請求アカウントとして AWS アカウント 指定された で行われます。ただし、その AWS アカウント または他の接続された で CodeCatalyst サービスロールを作成すると AWS アカウント、 CodeCatalyst サービスロールによって作成および使用されるリソースは、接続された で課金されます AWS アカウント。詳細については、「Amazon CodeCatalyst 管理者ガイド」の「請求の管理」を参照してください。

トピック

- IAM のアイデンティティベースのポリシー
- IAM のポリシーアクション
- IAM のポリシーリソース
- IAM のポリシー条件キー
- 接続の CodeCatalystアイデンティティベースのポリシーの例
- タグを使用してアカウント接続リソースへのアクセスを制御する

- CodeCatalyst アクセス許可リファレンス
- CodeCatalyst のサービスにリンクされたロールの使用
- AWS Amazon の マネージドポリシー CodeCatalyst
- IAM ロールを使用してプロジェクト AWS リソースへのアクセスを許可する

IAM のアイデンティティベースのポリシー

アイデンティティベースのポリシーは、アイデンティティにアタッチできる JSON アクセス許可ポリシードキュメントです。そのアイデンティティは、ユーザー、ユーザーのグループ、またはロールです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースのポリシーを作成する方法については、『IAM ユーザーガイド』の「IAM ポリシーの作成」を参照してください。

IAM アイデンティティベースのポリシーでは、許可または拒否するアクションとリソース、およびアクションを許可または拒否する条件を指定できます。プリンシパルは、それが添付されているユーザーまたはロールに適用されるため、アイデンティティベースのポリシーでは指定できません。JSON ポリシーで使用できるすべての要素については、「IAM ユーザーガイド」の「IAM JSON ポリシーの要素のリファレンス」を参照してください。

CodeCatalyst のアイデンティティベースのポリシーの例

CodeCatalyst アイデンティティベースのポリシーの例を表示するには、「」を参照してください接続の CodeCatalystアイデンティティベースのポリシーの例。

IAM のポリシーアクション

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソース に対してどのアクションを実行できるか、どの条件でどのアクションを実行できるかです。

JSON ポリシーの`Action`要素には、ポリシー内のアクセスを許可または拒否するために使用できるアクションが記述されます。ポリシーアクションの名前は通常、関連付けられた AWS API オペレーションと同じです。一致する API オペレーションのない権限のみのアクションなど、いくつかの例外があります。また、ポリシーに複数アクションが必要なオペレーションもあります。これらの追加アクションは、 依存アクション と呼ばれます。

単一のステートメントで複数のアクションを指定するには、アクションをカンマで区切ります。

```
"Action": [
      "prefix:action1",
      "prefix:action2"
         ]
```

IAM のポリシーリソース

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのアクションを実行できるか、どの条件でどのアクションを実行できるかです。

Resource JSON ポリシー要素は、アクションが適用されるオブジェクトを指定します。ステートメントには、Resource または NotResource要素を含める必要があります。ベストプラクティスとして、Amazon リソースネーム (ARN) を使用してリソースを指定します。これは、リソースレベルの権限と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルの権限をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

```
"Resource": "*"
```

IAM のポリシー条件キー

管理者は AWS JSON ポリシーを使用して、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソース に対してどのアクションを実行できるか、どの条件でどのアクションを実行できるかです。

Condition 要素 (または Condition ブロック) を使用すると、ステートメントが有効な条件を指定できます。Condition 要素はオプションです。イコールや未満などの 条件演算子 を使用して条件式を作成することで、ポリシーの条件とリクエスト内の値を一致させることができます。

1 つのステートメントに複数の Condition 要素を指定するか、1 つの Condition 要素に複数のキーを指定すると、 AWS は AND 論理演算子を使用してそれらを評価します。単一の条件キーに複数の値を指定すると、 AWS は OR 論理演算子を使用して条件を評価します。ステートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。詳細については、『IAM ユーザーガイド』の「IAM ポリシーの要素: 変数およびタグ」を参照してください。

AWS は、グローバル条件キーとサービス固有の条件キーをサポートします。すべての AWS グローバル条件キーを確認するには、『IAM ユーザーガイド』の「AWS グローバル条件コンテキストキー」を参照してください。

接続の CodeCatalystアイデンティティベースのポリシーの例

AWS アカウント では CodeCatalyst、スペースの請求を管理し、プロジェクトワークフローのリソースにアクセスするために が必要です。アカウント接続は、スペース AWS アカウント への追加を許可するために使用されます。アイデンティティベースのポリシーは、接続された で使用されます AWS アカウント。

デフォルトでは、ユーザーとロールにはリソースを作成または変更 CodeCatalystするアクセス許可はありません。また、、 AWS Command Line Interface （ AWS CLI ） AWS Management Console、または AWS API を使用してタスクを実行することはできません。IAM 管理者は、リソースで必要なアクションを実行するための許可をユーザーとロールに付与する IAM ポリシーを作成する必要があります。次に、管理者はこれらのポリシーを必要とするユーザーに、ポリシーをアタッチする必要があります。

次の IAM ポリシーの例では、アカウント接続に関連するアクションのアクセス許可を付与します。これらを使用して、アカウントを に接続するためのアクセスを制限します CodeCatalyst。

例 1: ユーザーが 1 つの で接続リクエストを受け入れることを許可する AWS リージョン

次のアクセス許可ポリシーでは、ユーザーが と 間の接続リクエストを表示 CodeCatalyst および承諾することのみを許可します AWS アカウント。さらに、ポリシーは 条件を使用して、us-west-2 リージョンのアクションのみを許可し、他の からのアクションは許可しません AWS リージョン。リクエストを表示して承認するには、ユーザーはリクエストで指定されたアカウントと同じアカウント AWS Management Console で にサインインします。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codecatalyst:AcceptConnection",
        "codecatalyst:GetPendingConnection"
      ],
      "Resource": "*",
      "Condition": {
```

```
        "StringEquals": {
          "aws:RequestedRegion": "us-west-2"
        }
      }
    }
  ]
}
```

例 2: コンソールで 1 つの の接続の管理を許可する AWS リージョン

次のアクセス許可ポリシーでは、ユーザーは 1 つのリージョン AWS アカウント で と 間の CodeCatalyst接続を管理できます。このポリシーは、 条件を使用して、us-west-2 リージョンのアクションのみを許可し、他の からのアクションは許可しません AWS リージョン。接続を作成したら、 の オプションを選択してCodeCatalystWorkflowDevelopmentRole-***spaceName***ロールを作成できます AWS Management Console。このポリシー例では、 `iam:PassRole`アクションの条件に のサービスプリンシパルが含まれます CodeCatalyst。そのアクセス権を持つロールのみが に作成されます AWS Management Console。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codecatalyst:*"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestedRegion": "us-west-2"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:CreatePolicy",
                "iam:AttachRolePolicy",
                "iam:ListRoles"
            ],
            "Resource": "*"
```

```
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "codecatalyst.amazonaws.com",
                        "codecatalyst-runner.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

例 3: 接続の管理を拒否する

次のアクセス許可ポリシーは、ユーザーが CodeCatalyst と 間の接続を管理する機能を拒否します AWS アカウント。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "codecatalyst:*"
            ],
            "Resource": "*"
        }
    ]
}
```

タグを使用してアカウント接続リソースへのアクセスを制御する

タグはリソースに添付することも、リクエストでタグ付けをサポートするサービスに渡すこともできます。ポリシー内のリソースにはタグを付けることができ、ポリシー内の一部のアクションにはタグを含めることができます。タグ付け条件キーには、aws:RequestTagaws:ResourceTagおよび条

件キーが含まれます。IAM ポリシーを作成するときに、タグ条件キーを使用して以下をコントロールできます。

- 接続リソースにすでに割り当てられているタグに基づいて、どのユーザーが接続リソースに対してアクションを実行できるか。
- どのタグをアクションのリクエストで渡すことができるか。
- リクエストで特定のタグキーを使用できるかどうか。

以下の例は、 CodeCatalystアカウント接続ユーザーのポリシーでタグ条件を指定する方法を示しています。条件キーの詳細については、IAM のポリシー条件キー を参照してください。

例 1: リクエスト内のタグに基づいてアクションを許可する

次のポリシーは、アカウント接続を承認する権限をユーザーに付与します。

これを行うには、リクエストに指定されているタグ `Project` の値が `ProjectA` である場合に、`AcceptConnection` アクションと `TagResource` アクションを許可します。(この `aws:RequestTag` 条件キーを使用して、IAM リクエストで渡すことができるタグをコントロールします)。`aws:TagKeys` 条件は、タグキーの大文字と小文字を区別します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codecatalyst:AcceptConnection",
        "codecatalyst:TagResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/Project": "ProjectA"
        },
        "ForAllValues:StringEquals": {
          "aws:TagKeys": ["Project"]
        }
      }
    }
  ]
}
```

例 2: リソースタグに基づいてアクションを許可する

次のポリシーは、アカウント接続リソースに対してアクションを実行したり、アカウント接続リソースに関する情報を取得したりする権限をユーザーに付与します。

そのために、Project接続にその値が付いたタグがある場合、ProjectA特定のアクションを許可します。(この aws:ResourceTag 条件キーを使用して、IAM リクエストで渡すことができるタグをコントロールします)。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codecatalyst:GetConnection",
        "codecatalyst:DeleteConnection",
        "codecatalyst:AssociateIamRoleToConnection",
        "codecatalyst:DisassociateIamRoleFromConnection",
        "codecatalyst:ListIamRolesForConnection",
        "codecatalyst:PutBillingAuthorization"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Project": "ProjectA"
        }
      }
    }
  ]
}
```

CodeCatalyst アクセス許可リファレンス

このセクションでは、 AWS アカウント に接続されている のアカウント接続リソースで使用されるアクションのアクセス許可リファレンスを提供します CodeCatalyst。次のセクションでは、アカウントの接続に関連するアクセス許可のみのアクションについて説明します。

アカウント接続に必要なアクセス許可

アカウント接続を使用するには、次のアクセス許可が必要です。

CodeCatalyst アカウント接続の アクセス許可	必要な許可	リソース
AcceptConnection	このアカウント CodeCatalyst をスペースに接続するリクエストを受け入れるために必要です。これは単なる IAM ポリシーのアクセス許可であり、API アクションではありません。	ポリシーの Resource 要素ではワイルドカード (*) のみがサポートされます。
AssociateIamRoleToConnection	IAM ロールをアカウント接続に関連付けるために必要です。これは単なる IAM ポリシーのアクセス許可であり、API アクションではありません。	arn:aws:codecatalyst:region: *account_ID* :/connections/ *connection_ID*
DeleteConnection	アカウント接続を削除するために必要です。これは単なる IAM ポリシーのアクセス許可であり、API アクションではありません。	arn:aws:codecatalyst:region: *account_ID* :/connections/ *connection_ID*
DisassociateIamRoleFromConnection	アカウント接続から IAM ロールの関連付けを解除するために必要です。これは単なる IAM ポリシーのアクセス許可であり、API アクションではありません。	arn:aws:codecatalyst:region: *account_ID* :/connections/ *connection_ID*
GetBillingAuthorization	アカウント接続の請求承認を記述するために必要です。これは単なる IAM ポリシーのアクセス許可であり、API アクションではありません。	arn:aws:codecatalyst:region: *account_ID* :/connections/ *connection_ID*

CodeCatalyst アカウント接続の アクセス許可	必要な許可	リソース
GetConnection	アカウント接続を取得するために必要です。これは単なる IAM ポリシーのアクセス許可であり、API アクションではありません。	arn:aws:codecatalyst:region: *account_ID* :/connections/ *connection_ID*
GetPendingConnection	このアカウント CodeCatalyst をスペースに接続するための保留中のリクエストを取得するために必要です。これは単なる IAM ポリシーのアクセス許可であり、API アクションではありません。	ポリシーの Resource 要素ではワイルドカード (*) のみがサポートされます。
ListConnections	保留中でないアカウント接続を一覧表示するために必要です。これは単なる IAM ポリシーのアクセス許可であり、API アクションではありません。	ポリシーの Resource 要素ではワイルドカード (*) のみがサポートされます。
ListIamRolesForConnection	アカウント接続に関連付けられた IAM ロールを一覧表示するために必要です。これは単なる IAM ポリシーのアクセス許可であり、API アクションではありません。	arn:aws:codecatalyst:region: *account_ID* :/connections/ *connection_ID*
ListTagsForResource	アカウント接続に関連付けられたタグを一覧表示するために必要です。これは単なる IAM ポリシーのアクセス許可であり、API アクションではありません。	arn:aws:codecatalyst:region: *account_ID* :/connections/ *connection_ID*

CodeCatalyst アカウント接続の アクセス許可	必要な許可	リソース
PutBillingAuthorization	アカウント接続の請求承認を作成または更新するために必要です。これは単なる IAM ポリシーのアクセス許可であり、API アクションではありません。	arn:aws:codecatalyst:region: *account_ID* :/connections/ *connection_ID*
RejectConnection	このアカウント CodeCatalystをスペースに接続するリクエストを拒否するために必要です。これは単なる IAM ポリシーのアクセス許可であり、API アクションではありません。	ポリシーの Resource 要素ではワイルドカード (*) のみがサポートされます。
TagResource	アカウント接続に関連付けられたタグを作成または編集するために必要です。これは単なる IAM ポリシーのアクセス許可であり、API アクションではありません。	arn:aws:codecatalyst:region: *account_ID* :/connections/ *connection_ID*
UntagResource	アカウント接続に関連付けられたタグを削除するために必要です。これは単なる IAM ポリシーのアクセス許可であり、API アクションではありません。	arn:aws:codecatalyst:region: *account_ID* :/connections/ *connection_ID*

IAM Identity Center アプリケーションに必要なアクセス許可

IAM Identity Center アプリケーションを使用するには、次のアクセス許可が必要です。

CodeCatalyst IAM Identity Center アプリケーションの アクセス許可	必要な許可	リソース
AssociateIdentityCenterApplicationToSpace	IAM Identity Center アプリケーションを CodeCatalyst スペースに関連付けるために必要です。これは単なる IAM ポリシーのアクセス許可であり、API アクションではありません。	arn:aws:codecatalyst:region: *account_ID* :/identity-center-applications/ *identity-center-application_ID*
AssociateIdentityToIdentityCenterApplication	スペースの IAM Identity Center アプリケーションに ID を関連付けるために CodeCatalyst必要です。これは単なる IAM ポリシーのアクセス許可であり、API アクションではありません。	arn:aws:codecatalyst:region: *account_ID* :/identity-center-applications/ *identity-center-application_ID*
BatchAssociateIdentitiesToIdentityCenterApplication	CodeCatalyst スペースの IAM Identity Center アプリケーションに複数の ID を関連付けるために必要です。これは単なる IAM ポリシーのアクセス許可であり、API アクションではありません。	arn:aws:codecatalyst:region: *account_ID* :/identity-center-applications/ *identity-center-application_ID*
BatchDisassociateIdentitiesFromIdentityCenterApplication	CodeCatalyst スペースの IAM Identity Center アプリケーションから複数の ID の関連付けを解除するために必要です。これは単なる IAM ポリシーのアクセス許可であり、API アクションではありません。	arn:aws:codecatalyst:region: *account_ID* :/identity-center-applications/ *identity-center-application_ID*

CodeCatalyst IAM Identity Center アプリケーションの アクセス許可	必要な許可	リソース
CreateIdentityCenterApplication	IAM Identity Center アプリケーションの作成に必要です。これは単なる IAM ポリシーのアクセス許可であり、API アクションではありません。	`arn:aws:codecatalyst:region: account_ID :/identity-center-applications/ identity-center-application_ID`
CreateSpaceAdminRoleAssignment	特定の CodeCatalystスペースと IAM Identity Center アプリケーションの管理者ロール割り当てを作成するために必要です。これは単なる IAM ポリシーのアクセス許可であり、API アクションではありません。	`arn:aws:codecatalyst:region: account_ID :/identity-center-applications/ identity-center-application_ID`
DeleteIdentityCenterApplication	IAM Identity Center アプリケーションを削除するために必要です。これは単なる IAM ポリシーのアクセス許可であり、API アクションではありません。	`arn:aws:codecatalyst:region: account_ID :/identity-center-applications/ identity-center-application_ID`
DisassociateIdentityCenterApplicationFromSpace	IAM Identity Center アプリケーションを CodeCatalyst スペースから関連付け解除するために必要です。これは単なる IAM ポリシーのアクセス許可であり、API アクションではありません。	`arn:aws:codecatalyst:region: account_ID :/identity-center-applications/ identity-center-application_ID`

CodeCatalyst IAM Identity Center アプリケーションの アクセス許可	必要な許可	リソース
DisassociateIdentityFromIdentityCenterApplication	CodeCatalyst スペースの IAM Identity Center アプリケーションから ID の関連付けを解除するために必要です。これは単なる IAM ポリシーのアクセス許可であり、API アクションではありません。	`arn:aws:codecatalyst:region: account_ID :/identity-center-applications/ identity-center-application_ID`
GetIdentityCenterApplication	IAM Identity Center アプリケーションに関する情報を取得するために必要です。これは単なる IAM ポリシーのアクセス許可であり、API アクションではありません。	`arn:aws:codecatalyst:region: account_ID :/identity-center-applications/ identity-center-application_ID`
ListIdentityCenterApplications	アカウント内のすべての IAM Identity Center アプリケーションのリストを表示するために必要です。これは単なる IAM ポリシーのアクセス許可であり、API アクションではありません。	ポリシーの `Resource` 要素ではワイルドカード (*) のみがサポートされます。
ListIdentityCenterApplicationsForSpace	IAM Identity Center アプリケーションのリストを CodeCatalyst スペース別に表示するために必要です。これは単なる IAM ポリシーのアクセス許可であり、API アクションではありません。	`arn:aws:codecatalyst:region: account_ID :/identity-center-applications/ identity-center-application_ID`

CodeCatalyst IAM Identity Center アプリケーションの アクセス許可	必要な許可	リソース
ListSpacesForIdentityCenterApplication	IAM Identity Center アプリケーションによって CodeCatalyst スペースのリストを表示するために必要です。これは単なる IAM ポリシーのアクセス許可であり、API アクションではありません。	`arn:aws:codecatalyst:region: account_ID :/identity-center-applications/ identity-center-application_ID`
SynchronizeIdentityCenterApplication	IAM Identity Center アプリケーションをバッキング ID ストアと同期させるのに必要です。これは単なる IAM ポリシーのアクセス許可であり、API アクションではありません。	`arn:aws:codecatalyst:region: account_ID :/identity-center-applications/ identity-center-application_ID`
UpdateIdentityCenterApplication	IAM Identity Center アプリケーションを更新するために必要です。これは単なる IAM ポリシーのアクセス許可であり、API アクションではありません。	`arn:aws:codecatalyst:region: account_ID :/identity-center-applications/ identity-center-application_ID`

CodeCatalyst のサービスにリンクされたロールの使用

Amazon CodeCatalyst は AWS Identity and Access Management (IAM) サービスにリンクされたロール を使用します。サービスにリンクされたロールは、 に直接リンクされた一意のタイプの IAM ロールです CodeCatalyst。サービスにリンクされたロールは、 によって事前定義 CodeCatalyst されており、ユーザーに代わってサービスから他の AWSのサービスを呼び出す必要のあるアクセス許可がすべて含まれています。

サービスにリンクされたロールを使用すると、必要なアクセス許可を手動で追加する必要がなくなるため、 の設定 CodeCatalyst が簡単になります。 は、サービスにリンクされたロールのアクセス許可 CodeCatalyst を定義し、特に定義されている場合を除き、 のみがそのロールを引き受け CodeCatalyst ることができます。定義された権限には、信頼ポリシーと権限ポリシーに含まれており、その権限ポリシーを他の IAM エンティティにアタッチすることはできません。

サービスリンクロールは、まずその関連リソースを削除しなければ削除できません。これにより、CodeCatalyst リソースへのアクセス許可を誤って削除することが防止され、リソースが保護されます。

サービスにリンクされたロールをサポートする他のサービスについては、「IAM と連動する AWS のサービス」を参照し、[Service-linked roles(サービスにリンクされたロール)] の列内で [Yes (はい)] と表記されたサービスを確認してください。そのサービスに関するサービスリンクロールのドキュメントを表示するには、リンクが設定されている [Yes (はい)] を選択します。

のサービスにリンクされたロールのアクセス許可 CodeCatalyst

CodeCatalyst は、 という名前のサービスにリンクされたロールを使用しますAmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronization。これにより、Amazon がユーザーに代わってアプリケーションインスタンスプロファイルおよび関連するディレクトリユーザーおよびグループへの CodeCatalyst 読み取り専用アクセスを許可します。

AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronization サービスにリンクされたロールは、ロールの引き受けについて以下のサービスを信頼します。

- `codecatalyst.amazonaws.com`

という名前のロールのアクセス許可ポリシー AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronizationPolicyは CodeCatalyst 、 が指定されたリソースに対して以下のアクションを実行することを許可します。

- アクション: `View application instance profiles and associated directory users and groups` の `CodeCatalyst spaces that support identity federation and SSO users and groups`

ユーザー、グループ、ロールなどがサービスにリンクされたロールを作成、編集、削除できるようにするには、アクセス権限を設定する必要があります。詳細については、 IAM ユーザーガイド の「サービスリンクロールのアクセス許可」を参照してください。

のサービスにリンクされたロールの作成 CodeCatalyst

サービスリンクロールを手動で作成する必要はありません。AWS Management Console、、AWS CLIまたは AWS API でスペースを作成すると、 によってサービスにリンクされたロール CodeCatalyst が作成されます。

> **Important**
>
> このサービスリンクロールは、このロールでサポートされている機能を使用する別のサービスでアクションが完了した場合にアカウントに表示されます。また、 CodeCatalyst サービスにリンクされたロールのサポートが開始された 2023 年 11 月 17 日より前に サービスを使用していた場合、 はアカウントに AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronizationロール CodeCatalyst を作成しました。詳細については、「AWS アカウント に新しいロールが表示される」を参照してください。

このサービスリンクロールを削除した後で再度作成する必要が生じた場合は、同じ方法でアカウントにロールを再作成できます。スペースを作成すると、 によってサービスにリンクされたロールが再度 CodeCatalyst 作成されます。

IAM コンソールを使用して、アプリケーションインスタンスプロファイルと関連するディレクトリユーザーおよびグループのユースケースでサービスにリンクされたロールを作成することもできます。AWS CLI または AWS API では、`codecatalyst.amazonaws.com` サービス名を使用してサービスにリンクされたロールを作成します。詳細については、『IAM ユーザーガイド』の「サービスにリンクされたロールの作成」を参照してください。このサービスにリンクされたロールを削除しても、この同じプロセスを使用して、もう一度ロールを作成できます。

のサービスにリンクされたロールの編集 CodeCatalyst

CodeCatalyst では、 AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronization サービスにリンクされたロールを編集することはできません。サービスリンクロールを作成すると、多くのエンティティによってロールが参照される可能性があるため、ロール名を変更することはできません。ただし、IAM を使用したロールの説明の編集はできます。詳細については、『IAM ユーザーガイド』の「サービスにリンクされたロールの編集」を参照してください。

のサービスにリンクされたロールの削除 CodeCatalyst

AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronization ロールを手動で削除する必要はありません。AWS Management Console、、または AWS API でスペースを削除する

と、 によってリソースが CodeCatalyst クリーンアップされAWS CLI、サービスにリンクされたロールが削除されます。

サービスリンクロールは、IAM コンソール、AWS CLI、または AWS API を使用して手動で削除することもできます。そのためにはまず、サービスリンクロールのリソースをクリーンアップする必要があります。その後で、手動で削除できます。

> **Note**
>
> リソースを削除する際に、 CodeCatalyst サービスでロールが使用されている場合、削除は失敗することがあります。失敗した場合は、数分待ってから操作を再試行してください。

で使用されている CodeCatalyst リソースを削除するには AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronization

- スペース を削除します。

IAM を使用してサービスリンクロールを手動で削除するには

IAM コンソール、AWS CLI、または AWS API を使用して、AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronization サービスリンクロールを削除します。詳細については、 IAM ユーザーガイド の「サービスにリンクされたロールの削除」を参照してください。

CodeCatalyst サービスにリンクされたロールをサポートするリージョン

CodeCatalyst は、このサービスを利用できるすべてのリージョンで、サービスにリンクされたロールの使用をサポートします。詳細については、「AWS リージョンとエンドポイント」を参照してください。

CodeCatalyst は、サービスを利用できるすべてのリージョンで、サービスにリンクされたロールの使用をサポートしているわけではありません。以下のリージョンでは、AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronization ロールを使用できます。

リージョン名	リージョン識別子	でのサポート CodeCatalyst
米国東部 (バージニア北部)	us-east-1	いいえ
米国東部 (オハイオ)	us-east-2	いいえ
米国西部 (北カリフォルニア)	us-west-1	いいえ
米国西部 (オレゴン)	us-west-2	はい
アフリカ (ケープタウン)	af-south-1	いいえ
アジアパシフィック (香港)	ap-east-1	いいえ
アジアパシフィック (ジャカルタ)	ap-southeast-3	いいえ
アジアパシフィック (ムンバイ)	ap-south-1	いいえ
アジアパシフィック (大阪)	ap-northeast-3	いいえ
アジアパシフィック (ソウル)	ap-northeast-2	いいえ
アジアパシフィック (シンガポール)	ap-southeast-1	いいえ
アジアパシフィック (シドニー)	ap-southeast-2	いいえ
アジアパシフィック (東京)	ap-northeast-1	いいえ
カナダ (中部)	ca-central-1	いいえ
欧州 (フランクフルト)	eu-central-1	いいえ
欧州 (アイルランド)	eu-west-1	はい
欧州 (ロンドン)	eu-west-2	いいえ
欧州 (ミラノ)	eu-south-1	いいえ
欧州 (パリ)	eu-west-3	いいえ
欧州 (ストックホルム)	eu-north-1	いいえ

リージョン名	リージョン識別子	でのサポート CodeCatalyst
中東 (バーレーン)	me-south-1	いいえ
中東 (アラブ首長国連邦)	me-central-1	いいえ
南米 (サンパウロ)	sa-east-1	いいえ
AWS GovCloud （米国東部）	us-gov-east-1	いいえ
AWS GovCloud （米国西部）	us-gov-west-1	いいえ

AWS Amazon の マネージドポリシー CodeCatalyst

AWS マネージドポリシーは、AWS が作成および管理するスタンドアロンポリシーです。AWS マネージドポリシーは、多くの一般的なユースケースで権限を提供できるように設計されているため、ユーザー、グループ、ロールへの権限の割り当てを開始できます。

AWS マネージドポリシーは、ご利用の特定のユースケースに対して最小特権の権限を付与しない場合があることにご注意ください。AWS のすべてのお客様が使用できるようになるのを避けるためです。ユースケース別にカスタマー管理ポリシーを定義することで、権限を絞り込むことをお勧めします。

AWS マネージドポリシーで定義したアクセス権限は変更できません。AWS が AWS マネージドポリシーに定義されているアクセス許可を更新すると、更新はポリシーがアタッチされているすべてのプリンシパルアイデンティティ 「ユーザー、グループ、ロール) に影響します。新しい AWS のサービスを起動するか、既存のサービスで新しい API オペレーションが使用可能になると、AWS が AWS マネージドポリシーを更新する可能性が最も高くなります。

詳細については、「IAM ユーザーガイド」の「AWS マネージドポリシー」を参照してください。

AWS マネージドポリシー: AmazonCodeCatalystSupportAccess

これは、すべてのスペース管理者とスペースメンバーが、スペース請求アカウントに関連付けられたビジネスまたはエンタープライズプレミアムサポートプランを利用するためのアクセス許可を付与するポリシーです。これらのアクセス許可により、スペース管理者とメンバーは、アクセス許可ポリシー内で CodeCatalyst アクセス許可を持つリソースのプレミアムサポートプランを利用できます。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- support – ユーザーが AWS サポートケースを検索、作成、解決できるようにするアクセス許可を付与します。また、コミュニケーション、重要度レベル、添付ファイル、および関連するサポートケースの詳細を記述するアクセス許可も付与します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "support:DescribeAttachment",
        "support:DescribeCaseAttributes",
        "support:DescribeCases",
        "support:DescribeCommunications",
        "support:DescribeIssueTypes",
        "support:DescribeServices",
        "support:DescribeSeverityLevels",
        "support:DescribeSupportLevel",
        "support:SearchForCases",
        "support:AddAttachmentsToSet",
        "support:AddCommunicationToCase",
        "support:CreateCase",
        "support:InitiateCallForCase",
        "support:InitiateChatForCase",
        "support:PutCaseAttributes",
        "support:RateCaseCommunication",
        "support:ResolveCase"
      ],
      "Resource": "*"
```

```
    }
  ]
}
```

AWS マネージドポリシー: AmazonCodeCatalystFullAccess

これは、 の Amazon CodeCatalyst Spaces ページで CodeCatalyst スペースと接続されたアカウントを管理するアクセス許可を付与するポリシーですAWS Management Console。このアプリケーションは、 のスペースAWS アカウントに接続されている を設定するために使用されますCodeCatalyst。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- `codecatalyst` – の Amazon CodeCatalyst Spaces ページに完全なアクセス許可を付与しますAWS Management Console。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CodeCatalystResourceAccess"
            "Effect": "Allow",
            "Action": [
                "codecatalyst:*",
                "iam:ListRoles"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CodeCatalystAssociateIAMRole"
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "*",
            "Condition": {
```

```
                "StringEquals": {
                    "iam:PassedToService": [
                        "codecatalyst.amazonaws.com",
                        "codecatalyst-runner.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

AWS マネージドポリシー: AmazonCodeCatalystReadOnlyAccess

これは、 の Amazon CodeCatalyst Spaces ページでスペースおよび接続されたアカウントの情報を表示および一覧表示するアクセス許可を付与するポリシーですAWS Management Console。このアプリケーションは、 のスペースAWS アカウントに接続されている を設定するために使用されますCodeCatalyst。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- `codecatalyst` – の Amazon CodeCatalyst Spaces ページに読み取り専用アクセス許可を付与しますAWS Management Console。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codecatalyst:Get*",
                "codecatalyst:List*",
            ],
            "Resource": "*"
        }
    ]
}
```

AWS マネージドポリシー:
AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronizationPolicy

AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronizationPolicyを IAM エンティティにアタッチすることはできません。このポリシーは、 がユーザーに代わって CodeCatalyst アクションを実行することを許可する、サービスにリンクされたロールにアタッチされます。詳細については、「 CodeCatalyst のサービスにリンクされたロールの使用」を参照してください。

このポリシーでは、 でスペースを管理するときに、アプリケーションインスタンスプロファイルと関連するディレクトリユーザーおよびグループを表示できます CodeCatalyst。ID フェデレーションと SSO ユーザーおよびグループをサポートするスペースを管理すると、お客様はこれらのリソースを表示します。

アクセス許可の詳細

このポリシーには、以下のアクセス許可が含まれています。

- sso – IAM Identity Center で管理されているアプリケーションインスタンスプロファイルを、 の関連付けられたスペースについて表示できるようにするアクセス許可を付与します CodeCatalyst。

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid":
 "AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronizationPolicy",
   "Effect": "Allow",
   "Action": [
    "sso:ListInstances",
    "sso:ListApplications",
    "sso:ListApplicationAssignments",
    "sso:DescribeInstance",
    "sso:DescribeApplication"
   ],
   "Resource": "*"
  }
```

```
 ]
}
```

AWS マネージドポリシーに関する CodeCatalyst の更新

このサービスがこれらの変更の追跡を開始 CodeCatalyst してからの の AWSマネージドポリシーの更新に関する詳細を表示します。このページの変更に関する自動通知を入手するには、CodeCatalyst ドキュメント履歴ページの RSS フィードをサブスクライブしてください。

変更	説明	日付
AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronizationPolicy - 新しいポリシー	CodeCatalyst が ポリシーを追加しました。 CodeCatalyst ユーザーがアプリケーションインスタンスプロファイルと関連するディレクトリユーザーおよびグループを表示できるようにするアクセス許可を付与します。	2023 年 11 月 17 日
AmazonCodeCatalystSupportAccess - 新しいポリシー	CodeCatalyst が ポリシーを追加しました。 CodeCatalyst ユーザーがサポートケースを検索、作成、解決したり、関連するコミュニケーションや詳細を表示したりできるようにするアクセス許可を付与します。	2023 年 4 月 20 日
AmazonCodeCatalystFullAccess - 新しいポリシー	CodeCatalyst が ポリシーを追加しました。 へのフルアクセスを付与します CodeCatalyst。	2023 年 4 月 20 日

変更	説明	日付
AmazonCodeCatalystReadOnlyAccess - 新しいポリシー	CodeCatalyst が ポリシーを追加しました。 への読み取り専用アクセスを許可します CodeCatalyst。	2023 年 4 月 20 日
CodeCatalyst が変更の追跡を開始	CodeCatalyst が AWSマネージドポリシーの変更の追跡を開始しました。	2023 年 4 月 20 日

IAM ロールを使用してプロジェクト AWS リソースへのアクセスを許可する

CodeCatalyst は、 を CodeCatalyst スペースに接続することで AWS アカウント AWS リソースにアクセスできます。その後、次のサービスロールを作成し、アカウントを接続するときに関連付けることができます。

JSON ポリシーで使用する要素の詳細については、「IAM ユーザーガイド」の「IAM JSON ポリシー要素リファレンス」を参照してください。

- CodeCatalyst プロジェクトとワークフロー AWS アカウント の のリソースにアクセスするには、まず、 がユーザーに代わってそれらのリソースにアクセス CodeCatalyst するためのアクセス許可を付与する必要があります。そのためには、 がスペース内のユーザーとプロジェクトに代わって引き受け AWS アカウント ることができる CodeCatalyst接続された にサービスロールを作成する必要があります。CodeCatalystWorkflowDevelopmentRole-*spaceName* サービスロールを作成して使用するか、カスタマイズされたサービスロールを作成してこれらの IAM ポリシーとロールを手動で設定するかを選択できます。ベストプラクティスとして、これらのロールに必要な最小限のアクセス許可を割り当てます。

> **Note**
>
> カスタマイズされたサービスロールには、 CodeCatalyst サービスプリンシパルが必要です。 CodeCatalyst サービスプリンシパルと信頼モデルの詳細については、「」を参照してください CodeCatalyst 信頼モデルを理解する。

- 接続された を介してスペースのサポートを管理するには AWS アカウント、 CodeCatalyst ユーザーがサポートにアクセスできるようにするAWSRoleForCodeCatalystSupportサービスロールを作成して使用することを選択できます。 CodeCatalyst スペースのサポートの詳細については、「」を参照してくださいAWS SupportAmazon 用 CodeCatalyst。

CodeCatalystWorkflowDevelopmentRole-*spaceName* サービスロールについて

接続された でリソースを作成およびアクセスするために CodeCatalyst が使用できるスペースの IAM ロールを追加できます AWS アカウント。これはサービスロール と呼ばれます。サービスロールを作成する最も簡単な方法は、スペースを作成するときにサービスロールを追加し、そのロールのCodeCatalystWorkflowDevelopmentRole-*spaceName*オプションを選択します。これにより、 がAdministratorAccessアタッチされたサービスロールが作成されるだけでなく、 がスペース内のプロジェクトでユーザーに代わってロールを引き受け CodeCatalyst ることを許可する信頼ポリシーも作成されます。サービスロールの範囲は、個々のプロジェクトではなく、スペースに限定されます。このロールの作成については、「アカウントとスペースのCodeCatalystWorkflowDevelopmentRole-*spaceName*ロールの作成」を参照してください。各アカウントのスペースごとに作成できるロールは 1 つだけです。

> **Note**
>
> このロールは開発アカウントでのみ使用が推奨され、 AdministratorAccess AWS 管理ポリシーを使用して、この で新しいポリシーとリソースを作成するためのフルアクセスを付与します AWS アカウント。

CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールにアタッチされたポリシーは、スペース内のブループリントで作成されたプロジェクトで動作するように設計されています。これにより、これらのプロジェクトのユーザーは、接続された のリソースを使用してコードを開発、構築、テスト、デプロイできます AWS アカウント。詳細については、「 AWS サービス用のロールの作成」を参照してください。

CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールにアタッチされたポリシーは、 の AdministratorAccessマネージドポリシーです AWS。これは、すべての AWS アクションとリソースへのフルアクセスを許可するポリシーです。IAM コンソールで JSON ポリシードキュメントを表示するには、「」を参照してくださいAdministratorAccess。

次の信頼ポリシーでは、 CodeCatalyst がCodeCatalystWorkflowDevelopmentRole-*spaceName*ロールを引き受けることを許可します。 CodeCatalyst 信頼モデルの詳細については、「」を参照してください CodeCatalyst 信頼モデルを理解する。

```
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
             "Principal": {
                "Service": [
                    "codecatalyst-runner.amazonaws.com",
                    "codecatalyst.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:codecatalyst:::space/spaceId/project/*"
                }
            }
        }
    ]
```

アカウントとスペースのCodeCatalystWorkflowDevelopmentRole-*spaceName*ロールの作成

スペース内のワークフローに使用するCodeCatalystWorkflowDevelopmentRole-*spaceName*ロールを作成するには、次の手順に従います。プロジェクトで使用する IAM ロールを持つアカウントごとに、スペースに開発者ロールなどのロールを追加する必要があります。

開始する前に、 の管理者権限を持っている AWS アカウント か、管理者と連携できる必要があります。 AWS アカウント および IAM ロールが で使用される方法の詳細については、 CodeCatalyst「」を参照してください接続された AWS リソースへのアクセスを許可する AWS アカウント。

を作成して追加するには CodeCatalyst CodeCatalystWorkflowDevelopmentRole-*spaceName*

1. CodeCatalyst コンソールで を開始する前に、 を開き AWS Management Console、 AWS アカウント スペースに対して同じ でログインしていることを確認します。
2. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。

3. CodeCatalyst スペースに移動します。[Settings (設定)]、[AWS アカウント] の順に選択します。
4. ロール AWS アカウント を作成する のリンクを選択します。AWS アカウント 詳細ページが表示されます。
5. からロールの管理を選択します AWS Management Console。

   で「Amazon CodeCatalyst スペースへの IAM ロールの追加」ページが開きます AWS Management Console。これは Amazon CodeCatalyst スペーススページです。ページにアクセスするには、ログインが必要な場合があります。
6. IAM で CodeCatalyst 開発管理者ロールの作成を選択します。このオプションは、開発ロールのアクセス許可ポリシーと信頼ポリシーを含むサービスロールを作成します。ロールには という名前が付けられますCodeCatalystWorkflowDevelopmentRole-*spaceName*。ロールとロールポリシーの詳細については、「」を参照してくださいいCodeCatalystWorkflowDevelopmentRole-***spaceName*** サービスロールについて。

   > **Note**
   >
   > このロールはデベロッパーアカウントでのみ使用が推奨され、`AdministratorAccess` AWS 管理ポリシーを使用して、この で新しいポリシーとリソースを作成するためのフルアクセスを付与します AWS アカウント。
7. 開発ロールの作成 を選択します。
8. 接続ページの で使用できる IAM ロールで CodeCatalyst、アカウントに追加された IAM `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`*ロールのリストでロールを表示します。
9. スペースに戻るには、「Amazon に移動 CodeCatalyst」を選択します。

AWSRoleForCodeCatalystSupport サービスロールについて

スペースの CodeCatalyst ユーザーがサポートケースの作成とアクセスに使用できるスペースの IAM ロールを追加できます。これはサポート用のサービスロールと呼ばれます。サポート用のサービスロールを作成する最も簡単な方法は、スペースを作成するときにサービスロールを追加し、そのロール`AWSRoleForCodeCatalystSupport`のオプションを選択することです。これにより、ポリシーとロールが作成されるだけでなく、 がスペース内のプロジェクトでユーザーに代わってロールを引き受けること CodeCatalyst を許可する信頼ポリシーも作成されます。サービスロールの範囲は、個々のプロジェクトではなく、スペースに限定されます。このロールの作成については、「アカウントとスペースのAWSRoleForCodeCatalystSupportロールの作成」を参照してください。

AWSRoleForCodeCatalystSupport ロールにアタッチされたポリシーは、サポートアクセス許可へのアクセスを提供する管理ポリシーです。詳細については、「AWS マネージドポリシー: AmazonCodeCatalystSupportAccess」を参照してください。

ポリシーの信頼ロールは、 CodeCatalyst がロールを引き受けることを許可します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "codecatalyst.amazonaws.com",
                    "codecatalyst-runner.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

アカウントとスペースのAWSRoleForCodeCatalystSupportロールの作成

以下の手順に従って、スペース内のサポートケースに使用されるAWSRoleForCodeCatalystSupportロールを作成します。ロールは、スペースの指定された請求アカウントに追加する必要があります。

開始する前に、 の管理者権限を持っている AWS アカウント か、管理者と連携できる必要があります。 AWS アカウント および IAM ロールが で使用される方法の詳細については、 CodeCatalyst「」を参照してください接続された AWS リソースへのアクセスを許可する AWS アカウント。

を作成して追加するには CodeCatalyst AWSRoleForCodeCatalystSupport

1. CodeCatalyst コンソールで を開始する前に、 を開き AWS Management Console、 AWS アカウント スペースに対して同じ でログインしていることを確認します。
2. CodeCatalyst スペースに移動します。[Settings (設定)]、[AWS アカウント] の順に選択します。
3. ロール AWS アカウント を作成する のリンクを選択します。AWS アカウント 詳細ページが表示されます。
4. からロールの管理を選択します AWS Management Console。

で「Amazon CodeCatalyst スペースへの IAM ロールの追加」ページが開きます AWS Management Console。これは Amazon CodeCatalyst Spaces ページです。ページにアクセスするには、サインインが必要な場合があります。

5. CodeCatalyst スペースの詳細 で、 CodeCatalyst サポートロールの追加 を選択します。このオプションは、プレビュー開発ロールのアクセス許可ポリシーと信頼ポリシーを含むサービスロールを作成します。ロールには一意の識別子AWSRoleForCodeCatalystSupportが付加された名前が付けられます。ロールとロールポリシーの詳細については、「」を参照してくださいAWSRoleForCodeCatalystSupport サービスロールについて。
6. CodeCatalyst サポート用のロールを追加ページで、デフォルトを選択したまま、ロールの作成を選択します。
7. で使用できる IAM ロール CodeCatalystで、アカウントに追加された IAM `CodeCatalystWorkflowDevelopmentRole-`*`spaceName`*ロールのリストでロールを表示します。
8. スペースに戻るには、「Amazon に移動 CodeCatalyst」を選択します。

でのワークフローアクションの IAM ロールの設定 CodeCatalyst

このセクションでは、アカウントで使用できる IAM ロールとポリシーについて詳しく説明します CodeCatalyst。サンプルロールを作成する手順については、「」を参照してくださいワークフローアクション用のロールの手動作成。IAM ロールを作成したら、ロール ARN をコピーして IAM ロールをアカウント接続に追加し、プロジェクト環境に関連付けます。詳細については、「アカウント接続への IAM ロールの追加」を参照してください。

CodeCatalyst Amazon S3 アクセス用の ロールの構築

CodeCatalyst ワークフロービルドアクションでは、デフォルトのCodeCatalystWorkflowDevelopmentRole-***spaceName***サービスロールを使用するか、CodeCatalystBuildRoleforS3Access という名前の IAM ロールを作成できます。このロールは、の AWS CloudFormation リソースでタスクを実行する CodeCatalyst ために必要なスコープ付きアクセス許可を持つポリシーを使用します AWS アカウント。

このロールは、次のことを行うためのアクセス許可を付与します。

- Amazon S3 バケットに書き込みます。
- を使用した リソースの構築をサポートします AWS CloudFormation。これには Amazon S3 アクセスが必要です。

このロールは、次のポリシーを使用します。

```
{
    "Version": "2012-10-17",
    "Statement": [{
    "Action": [
        "s3:PutObject",
        "iam:PassRole"
    ],
    "Resource": "resource_ARN",
    "Effect": "Allow"
}]
}
```

> **Note**
>
> ロールを初めて使用してワークフローアクションを実行するときは、リソースポリシーステートメントでワイルドカードを使用し、使用可能になった後にリソース名でポリシーの範囲を絞り込みます。
>
> ```
> "Resource": "*"
> ```

CodeCatalyst の ロールの構築 AWS CloudFormation

CodeCatalyst ワークフロービルドアクションでは、デフォルトのCodeCatalystWorkflowDevelopmentRole-*spaceName*サービスロールを使用するか、必要なアクセス許可を持つ IAM ロールを作成できます。このロールは、 の AWS CloudFormation リソースでタスクを実行する CodeCatalyst ために必要なスコープ付きアクセス許可を持つポリシーを使用しますAWS アカウント。

このロールは、次のことを行うためのアクセス許可を付与します。

- を使用した リソースの構築をサポートします AWS CloudFormation。これは、Amazon S3 アクセスのビルドロールと の CodeCatalyst デプロイロールとともに CodeCatalyst必要です AWS CloudFormation。

このロールには、次の AWS マネージドポリシーをアタッチする必要があります。

- AWSCloudFormationFullAccess
- IAMFullAccess
- AmazonS3FullAccess
- AmazonAPIGatewayAdministrator
- AWSLambdaFullAccess

CodeCatalyst CDK の ロールの構築

Modern 3 層ウェブアプリケーションなどの CDK ビルドアクションを実行する CodeCatalyst ワークフローでは、デフォルトのCodeCatalystWorkflowDevelopmentRole-*spaceName*サービスロールを使用するか、必要なアクセス許可を持つ IAM ロールを作成できます。このロールは、 の AWS CloudFormation リソースの CDK ビルドコマンドをブートストラップして実行 CodeCatalyst するために必要な、スコープ付きアクセス許可を持つポリシーを使用します AWS アカウント。

このロールは、次のことを行うためのアクセス許可を付与します。

- Amazon S3 バケットに書き込みます。
- CDK コンストラクトと AWS CloudFormation リソーススタックの構築をサポートします。これには、アーティファクトストレージ用の Amazon S3、イメージリポジトリのサポート用の Amazon ECR、仮想インスタンスのシステムガバナンスとモニタリング用の SSM へのアクセスが必要です。

このロールは、次のポリシーを使用します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudformation:*",
                "ecr:*",
                "ssm:*",
                "s3:*",
                "iam:PassRole",
                "iam:GetRole",
                "iam:CreateRole",
                "iam:AttachRolePolicy",
                "iam:PutRolePolicy"
```

```
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> ロールを初めて使用してワークフローアクションを実行するときは、リソースポリシーステートメントでワイルドカードを使用し、使用可能になった後にリソース名でポリシーの範囲を絞り込みます。
>
> ```
> "Resource": "*"
> ```

CodeCatalyst のロールをデプロイする AWS CloudFormation

を使用する CodeCatalyst ワークフローデプロイアクションでは AWS CloudFormation、デフォルトのCodeCatalystWorkflowDevelopmentRole-*spaceName*サービスロールを使用するか、 の AWS CloudFormation リソースでタスクを実行する CodeCatalyst ために必要なスコープ付きアクセス許可を持つポリシーを使用できます AWS アカウント。

このロールは、次のことを行うためのアクセス許可を付与します。

- CodeCatalyst が を通じてブルー/グリーンデプロイを実行するために Λ 関数を呼び出すことを許可します AWS CloudFormation。
- CodeCatalyst でスタックと変更セットの作成と更新を に許可します AWS CloudFormation。

このロールは、次のポリシーを使用します。

```
{"Action": [
        "cloudformation:CreateStack",
        "cloudformation:DeleteStack",
        "cloudformation:Describe*",
        "cloudformation:UpdateStack",
        "cloudformation:CreateChangeSet",
        "cloudformation:DeleteChangeSet",
        "cloudformation:ExecuteChangeSet",
        "cloudformation:SetStackPolicy",
```

```
            "cloudformation:ValidateTemplate",
            "cloudformation:List*",
            "iam:PassRole"
        ],
        "Resource": "resource_ARN",
        "Effect": "Allow"
}
```

> **Note**
>
> ロールを初めて使用してワークフローアクションを実行するときは、リソースポリシーステートメントでワイルドカードを使用し、使用可能になった後にリソース名でポリシーの範囲を絞り込みます。
>
> ```
> "Resource": "*"
> ```

CodeCatalyst Amazon EC2 のロールをデプロイする

CodeCatalyst ワークフローデプロイアクションは、必要なアクセス許可を持つ IAM ロールを使用します。このロールは、 の Amazon EC2 リソースでタスクを実行する CodeCatalyst ために必要なスコープ付きアクセス許可を持つポリシーを使用します AWS アカウント。CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールのデフォルトポリシーには、Amazon EC2 または Amazon EC2 Auto Scaling のアクセス許可は含まれません。

このロールは、次のことを行うためのアクセス許可を付与します。

- Amazon EC2 デプロイを作成します。
- インスタンスのタグを読み取るか、Auto Scaling グループ名で Amazon EC2 インスタンスを識別します。
- Amazon EC2 Auto Scaling グループ、ライフサイクルフック、スケーリングポリシーの読み取り、作成、更新、削除を行います。
- Amazon SNS トピックに情報を公開します。
- CloudWatch アラームに関する情報を取得します。
- Elastic Load Balancing を読み、更新します。

このロールは、次のポリシーを使用します。

```
{
"Version": "2012-10-17",
"Statement": [
{
"Effect": "Allow",
"Action": [
 "autoscaling:CompleteLifecycleAction",
"autoscaling:DeleteLifecycleHook",
"autoscaling:DescribeAutoScalingGroups",
"autoscaling:DescribeLifecycleHooks",
"autoscaling:PutLifecycleHook",
"autoscaling:RecordLifecycleActionHeartbeat",
"autoscaling:CreateAutoScalingGroup",
"autoscaling:UpdateAutoScalingGroup",
"autoscaling:EnableMetricsCollection",
"autoscaling:DescribePolicies",
"autoscaling:DescribeScheduledActions",
"autoscaling:DescribeNotificationConfigurations",
"autoscaling:SuspendProcesses",
"autoscaling:ResumeProcesses",
"autoscaling:AttachLoadBalancers",
"autoscaling:AttachLoadBalancerTargetGroups",
"autoscaling:PutScalingPolicy",
"autoscaling:PutScheduledUpdateGroupAction",
"autoscaling:PutNotificationConfiguration",
"autoscaling:PutWarmPool",
"autoscaling:DescribeScalingActivities",
"autoscaling:DeleteAutoScalingGroup",
"ec2:DescribeInstances",
"ec2:DescribeInstanceStatus",
"ec2:TerminateInstances",
"tag:GetResources",
"sns:Publish",
"cloudwatch:DescribeAlarms",
"cloudwatch:PutMetricAlarm",
"elasticloadbalancing:DescribeLoadBalancers",
"elasticloadbalancing:DescribeInstanceHealth",
"elasticloadbalancing:RegisterInstancesWithLoadBalancer",
"elasticloadbalancing:DeregisterInstancesFromLoadBalancer",
"elasticloadbalancing:DescribeTargetGroups",
"elasticloadbalancing:DescribeTargetHealth",
"elasticloadbalancing:RegisterTargets",
"elasticloadbalancing:DeregisterTargets"
```

```
],
"Resource": "resource_ARN"
      }
   ]
}
```

> **Note**
>
> ロールを初めて使用してワークフローアクションを実行するときは、リソースポリシーステートメントでワイルドカードを使用し、使用可能になった後にリソース名でポリシーの範囲を絞り込みます。
>
> ```
> "Resource": "*"
> ```

CodeCatalyst Amazon ECS の ロールのデプロイ

CodeCatalyst ワークフローアクションでは、必要なアクセス許可を持つ IAM ロールを作成できます。デフォルトのCodeCatalystWorkflowDevelopmentRole-*spaceName*サービスロールを使用することも、Lambda CodeCatalyst デプロイに使用するデプロイアクション用の IAM ロールを作成することもできます。このロールは、 の Amazon ECS リソースでタスクを実行する CodeCatalyst ために必要なスコープ付きアクセス許可を持つポリシーを使用します AWS アカウント。

このロールは、次のことを行うためのアクセス許可を付与します。

- CodeCatalyst 接続で指定されたアカウントで、 CodeCatalyst ユーザーに代わって Amazon ECS のローリングデプロイを開始します。
- Amazon ECS タスクセットを読んで、更新、削除します。
- Elastic Load Balancing ターゲットグループ、リスナー、ルールを更新します。
- Lambda 関数を呼び出します。
- Amazon S3 バケットのリビジョンファイルにアクセスします。
- CloudWatch アラームに関する情報を取得します。
- Amazon SNS トピックに情報を公開します。

このロールは、次のポリシーを使用します。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [{
    "Action":[
      "ecs:DescribeServices",
      "ecs:CreateTaskSet",
      "ecs:DeleteTaskSet",
      "ecs:ListClusters",
      "ecs:RegisterTaskDefinition",
      "ecs:UpdateServicePrimaryTaskSet",
      "ecs:UpdateService",
      "elasticloadbalancing:DescribeTargetGroups",
      "elasticloadbalancing:DescribeListeners",
      "elasticloadbalancing:ModifyListener",
      "elasticloadbalancing:DescribeRules",
      "elasticloadbalancing:ModifyRule",
      "lambda:InvokeFunction",
      "lambda:ListFunctions",
      "cloudwatch:DescribeAlarms",
      "sns:Publish",
      "sns:ListTopics",
      "s3:GetObject",
      "s3:GetObjectVersion",
      "codedeploy:CreateApplication",
      "codedeploy:CreateDeployment",
      "codedeploy:CreateDeploymentGroup",
      "codedeploy:GetApplication",
      "codedeploy:GetDeployment",
      "codedeploy:GetDeploymentGroup",
      "codedeploy:ListApplications",
      "codedeploy:ListDeploymentGroups",
      "codedeploy:ListDeployments",
      "codedeploy:StopDeployment",
      "codedeploy:GetDeploymentTarget",
      "codedeploy:ListDeploymentTargets",
      "codedeploy:GetDeploymentConfig",
      "codedeploy:GetApplicationRevision",
      "codedeploy:RegisterApplicationRevision",
      "codedeploy:BatchGetApplicationRevisions",
      "codedeploy:BatchGetDeploymentGroups",
      "codedeploy:BatchGetDeployments",
      "codedeploy:BatchGetApplications",
      "codedeploy:ListApplicationRevisions",
      "codedeploy:ListDeploymentConfigs",
      "codedeploy:ContinueDeployment"
```

```
    ],
    "Resource":"*",
    "Effect":"Allow"
},{"Action":[
       "iam:PassRole"
    ],
    "Effect":"Allow",
    "Resource":"*",
    "Condition":{"StringLike":{"iam:PassedToService":[
             "ecs-tasks.amazonaws.com",
             "codedeploy.amazonaws.com"
          ]
       }
    }
}]
}
```

> **Note**
>
> ロールを初めて使用してワークフローアクションを実行するときは、リソースポリシーステートメントでワイルドカードを使用し、使用可能になった後にリソース名でポリシーの範囲を絞り込みます。
>
> ```
> "Resource": "*"
> ```

CodeCatalyst Lambda のロールをデプロイする

CodeCatalyst ワークフローアクションでは、必要なアクセス許可を持つ IAM ロールを作成できます。デフォルトのCodeCatalystWorkflowDevelopmentRole-*spaceName*サービスロールを使用するか、Lambda デプロイに使用するデプロイアクション用の CodeCatalyst IAM ロールを作成できます。このロールは、 の Lambda リソースでタスクを実行する CodeCatalyst ために必要なスコープ付きアクセス許可を持つポリシーを使用します AWS アカウント。

このロールは、次のことを行うためのアクセス許可を付与します。

- Lambda 関数とエイリアスの読み取り、更新、呼び出しを行います。
- Amazon S3 バケットのリビジョンファイルにアクセスします。
- CloudWatch イベントアラームに関する情報を取得します。
- Amazon SNS トピックに情報を公開します。

このロールは、次のポリシーを使用します。

```
*{*
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "cloudwatch:DescribeAlarms",
                "lambda:UpdateAlias",
                "lambda:GetAlias",
                "lambda:GetProvisionedConcurrencyConfig",
                "sns:Publish"
            ],
            "Resource": "resource_ARN",
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": "arn:aws:s3:::/CodeDeploy/",
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": "",
            "Condition": {
                "StringEquals": {
                    "s3:ExistingObjectTag/UseWithCodeDeploy": "true"
                }
            },
            "Effect": "Allow"
        },
        {
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:::function:CodeDeployHook_*",
            "Effect": "Allow"
        }
```

```
    ]
}
```

> **Note**
>
> ロールを初めて使用してワークフローアクションを実行するときは、リソースポリシーステートメントでワイルドカードを使用し、使用可能になった後にリソース名でポリシーの範囲を絞り込みます。
>
> ```
> "Resource": "*"
> ```

CodeCatalyst Lambda のロールをデプロイする

CodeCatalyst ワークフローアクションでは、デフォルトのCodeCatalystWorkflowDevelopmentRole-***spaceName***サービスロールを使用するか、必要なアクセス許可を持つ IAM ロールを作成できます。このロールは、 の Lambda リソースでタスクを実行する CodeCatalyst ために必要なスコープ付きアクセス許可を持つポリシーを使用します AWS アカウント。

このロールは、次のことを行うためのアクセス許可を付与します。

- Lambda 関数とエイリアスの読み取り、更新、呼び出しを行います。
- Amazon S3 バケットのリビジョンファイルにアクセスします。
- CloudWatch アラームに関する情報を取得します。
- Amazon SNS トピックに情報を公開します。

このロールは、次のポリシーを使用します。

```
*{*
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "cloudwatch:DescribeAlarms",
                "lambda:UpdateAlias",
                "lambda:GetAlias",
                "lambda:GetProvisionedConcurrencyConfig",
```

```
                "sns:Publish"
            ],
            "Resource": "resource_ARN",
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": "arn:aws:s3:::/CodeDeploy/",
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": "",
            "Condition": {
                "StringEquals": {
                    "s3:ExistingObjectTag/UseWithCodeDeploy": "true"
                }
            },
            "Effect": "Allow"
        },
        {
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "arn:aws:lambda:::function:CodeDeployHook_*",
            "Effect": "Allow"
        }
    ]
}
```

> **Note**
>
> ロールを初めて使用してワークフローアクションを実行するときは、リソースポリシーステートメントでワイルドカードを使用し、使用可能になった後にリソース名でポリシーの範囲を絞り込みます。

```
        "Resource": "*"
```

CodeCatalyst のロールをデプロイする AWS SAM

CodeCatalyst ワークフローアクションでは、デフォルトのCodeCatalystWorkflowDevelopmentRole-*spaceName*サービスロールを使用するか、必要なアクセス許可を持つ IAM ロールを作成できます。このロールは、 の AWS SAM および AWS CloudFormation リソースでタスクを実行する CodeCatalyst ために必要なスコープ付きアクセス許可を持つポリシーを使用します AWS アカウント。

このロールは、次のことを行うためのアクセス許可を付与します。

- CodeCatalyst が Lambda 関数を呼び出して、サーバーレスおよび AWS SAM CLI アプリケーションのデプロイを実行できるようにします。
- CodeCatalyst でスタックと変更セットの作成と更新を に許可します AWS CloudFormation。

このロールは、次のポリシーを使用します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "iam:PassRole",
                "iam:DeleteRole",
                "iam:GetRole",
                "iam:TagRole",
                "iam:CreateRole",
                "iam:AttachRolePolicy",
                "iam:DetachRolePolicy",
                "cloudformation:*",
                "lambda:*",
                "apigateway:*"
            ],
            "Resource": "*"
        }
```

```
    ]
}
```

> **Note**
>
> ロールを初めて使用してワークフローアクションを実行するときは、リソースポリシーステートメントでワイルドカードを使用し、使用可能になった後にリソース名でポリシーの範囲を絞り込みます。
>
> ```
> "Resource": "*"
> ```

CodeCatalyst Amazon EC2 の読み取り専用ロール

CodeCatalyst ワークフローアクションでは、必要なアクセス許可を持つ IAM ロールを作成できます。このロールは、 の Amazon EC2 リソースでタスクを実行する CodeCatalyst ために必要なスコープ付きアクセス許可を持つポリシーを使用します AWS アカウント。CodeCatalystWorkflowDevelopmentRole-*spaceName* サービスロールには、Amazon EC2 のアクセス許可や、Amazon の説明されたアクションは含まれません CloudWatch。

このロールは、次のことを行うためのアクセス許可を付与します。

- Amazon EC2 インスタンスのステータスを取得します。
- Amazon EC2 インスタンスの CloudWatch メトリクスを取得します。

このロールは、次のポリシーを使用します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ec2:Describe",
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": "elasticloadbalancing:Describe",
            "Resource": "resource_ARN"
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:ListMetrics",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch:Describe"
            ],
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": "autoscaling:Describe",
            "Resource": "resource_ARN"
        }
    ]
}
```

> **Note**
>
> ロールを初めて使用してワークフローアクションを実行するときは、リソースポリシーステートメントでワイルドカードを使用し、使用可能になった後にリソース名でポリシーの範囲を絞り込みます。
>
> ```
> "Resource": "*"
> ```

CodeCatalyst Amazon ECS の読み取り専用ロール

CodeCatalyst ワークフローアクションでは、必要なアクセス許可を持つ IAM ロールを作成できます。このロールは、 の Amazon ECS リソースでタスクを実行する CodeCatalyst ために必要なスコープ付きアクセス許可を持つポリシーを使用します AWS アカウント。

このロールは、次のことを行うためのアクセス許可を付与します。

- Amazon ECS タスクセットを読み取ります。
- CloudWatch アラームに関する情報を取得します。

このロールは、次のポリシーを使用します。

```
*{*
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "ecs:DescribeServices",
                "cloudwatch:DescribeAlarms"
            ],
            "Resource": "resource_ARN",
            "Effect": "Allow"
        },
       {
            "Action": [
                "elasticloadbalancing:DescribeTargetGroups",
                "elasticloadbalancing:DescribeListeners",
                "elasticloadbalancing:DescribeRules"
            ],
            "Resource": "resource_ARN",
            "Effect": "Allow"
        },
       {
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": "",
            "Condition": {
                "StringEquals": {
                    "s3:ExistingObjectTag/UseWithCodeDeploy": "true"
                }
            },
            "Effect": "Allow"
        },
        {
            "Action": [
                "iam:PassRole"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iam:::role/ecsTaskExecutionRole",
                "arn:aws:iam:::role/ECSTaskExecution"
            ],
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": [
```

```
                        "ecs-tasks.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

> **Note**
>
> ロールを初めて使用してワークフローアクションを実行するときは、リソースポリシーステートメントでワイルドカードを使用し、使用可能になった後にリソース名でポリシーの範囲を絞り込みます。
>
> ```
> "Resource": "*"
> ```

CodeCatalyst Lambda の読み取り専用ロール

CodeCatalyst ワークフローアクションでは、必要なアクセス許可を持つ IAM ロールを作成できます。このロールは、 の Lambda リソースでタスクを実行する CodeCatalyst ために必要なスコープ付きアクセス許可を持つポリシーを使用します AWS アカウント。

このロールは、以下のアクセス許可を付与します。

- Lambda 関数とエイリアスを読み取ります。
- Amazon S3 バケットのリビジョンファイルにアクセスします。
- CloudWatch アラームに関する情報を取得します。

このロールは以下の ポリシーを使用します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "cloudwatch:DescribeAlarms",
                "lambda:GetAlias",
                "lambda:GetProvisionedConcurrencyConfig"
            ],
```

```
            "Resource": "resource_ARN",
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": "arn:aws:s3:::/CodeDeploy/",
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": "",
            "Condition": {
                "StringEquals": {
                    "s3:ExistingObjectTag/UseWithCodeDeploy": "true"
                }
            },
            "Effect": "Allow"
        }
    ]
}
```

> **Note**
>
> ロールを初めて使用してワークフローアクションを実行するときは、リソースポリシーステートメントでワイルドカードを使用し、使用可能になった後にリソース名でポリシーの範囲を絞り込みます。
>
> ```
> "Resource": "*"
> ```

ワークフローアクション用のロールの手動作成

CodeCatalyst ワークフローアクションは、ビルドロール 、デプロイロール 、スタックロール と呼ばれる、作成した IAM ロールを使用します。

これらのロールを IAM で作成するには、次の手順に従います。

デプロイロールを作成するには

1. 次のように、ロールのポリシーを作成します。

   a. にサインインします AWS。

   b. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。

   c. ナビゲーションペインで、ポリシー を選択します。

   d. ポリシーの作成を選択します。

   e. [JSON] タブを選択します。

   f. 既存のコードを削除します。

   g. 次のコードを貼り付けます。

```
{
    "Version": "2012-10-17",
    "Statement": [{
    "Action": [
        "cloudformation:CreateStack",
        "cloudformation:DeleteStack",
        "cloudformation:Describe*",
        "cloudformation:UpdateStack",
        "cloudformation:CreateChangeSet",
        "cloudformation:DeleteChangeSet",
        "cloudformation:ExecuteChangeSet",
        "cloudformation:SetStackPolicy",
        "cloudformation:ValidateTemplate",
        "cloudformation:List*",
        "iam:PassRole"
    ],
    "Resource": "*",
    "Effect": "Allow"
}]
}
```

> **Note**
>
> ロールを初めて使用してワークフローアクションを実行するときは、リソースポリシーステートメントでワイルドカードを使用し、使用可能になった後にリソース名でポリシーの範囲を絞り込みます。

```
        "Resource": "*"
```

h. [次へ : タグ] を選択します。

i. [次へ: レビュー] を選択します。

j. 名前 で、次のように入力します。

```
codecatalyst-deploy-policy
```

k. [ポリシーの作成] を選択します。

これで、アクセス許可ポリシーが作成されました。

2. デプロイロールを次のように作成します。

a. ナビゲーションペインで ロール を選択してから、ロールを作成する を選択します。

b. カスタム信頼ポリシー を選択します。

c. 既存のカスタム信頼ポリシーを削除します。

d. 次のカスタム信頼ポリシーを追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

e. [次へ] をクリックします。

f. アクセス許可ポリシー で、そのチェックボックスを検索codecatalyst-deploy-policyして選択します。

g. [次へ] をクリックします。

h. ロール名 には、次のように入力します。

```
codecatalyst-deploy-role
```

i. ロールの説明 には、次のように入力します。

```
CodeCatalyst deploy role
```

j. [ロールを作成] を選択します。

これで、信頼ポリシーとアクセス許可ポリシーを使用してデプロイロールが作成されました。

3. 次のように、デプロイロール ARN を取得します。

a. ナビゲーションペインで、[ロール] を選択します。

b. 検索ボックスに、作成したロールの名前 () を入力しますcodecatalyst-deploy-role。

c. リストからロールを選択します。

ロールの概要ページが表示されます。

d. 上部で、ARN 値をコピーします。

これで、適切なアクセス許可を持つデプロイロールを作成し、その ARN を取得しました。

ビルドロールを作成するには

1. 次のように、ロールのポリシーを作成します。

a. にサインインします AWS。

b. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。

c. ナビゲーションペインで、ポリシー を選択します。

d. ポリシーの作成を選択します。

e. [JSON] タブを選択します。

f. 既存のコードを削除します。

g. 次のコードを貼り付けます。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [{
    "Action": [
        "s3:PutObject",
        "iam:PassRole"
    ],
    "Resource": "*",
    "Effect": "Allow"
}]
}
```

> **Note**
>
> ロールを初めて使用してワークフローアクションを実行するときは、リソースポリシーステートメントでワイルドカードを使用し、使用可能になった後にリソース名でポリシーの範囲を絞り込みます。
>
> ```
> "Resource": "*"
> ```

h. [次へ : タグ] を選択します。

i. [次へ: レビュー] を選択します。

j. 名前 で、次のように入力します。

```
codecatalyst-build-policy
```

k. [ポリシーの作成] を選択します。

これで、アクセス許可ポリシーが作成されました。

2. 次のようにビルドロールを作成します。

a. ナビゲーションペインで ロール を選択してから、ロールを作成する を選択します。

b. カスタム信頼ポリシー を選択します。

c. 既存のカスタム信頼ポリシーを削除します。

d. 次のカスタム信頼ポリシーを追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

e. [次へ] をクリックします。

f. アクセス許可ポリシー で、そのチェックボックスを検索codecatalyst-build-policyして選択します。

g. [次へ] をクリックします。

h. ロール名 には、次のように入力します。

```
codecatalyst-build-role
```

i. ロールの説明 には、次のように入力します。

```
CodeCatalyst build role
```

j. [ロールを作成] を選択します。

これで、信頼ポリシーとアクセス許可ポリシーを使用してビルドロールが作成されました。

3. 次のように、ビルドロール ARN を取得します。

a. ナビゲーションペインで、[ロール] を選択します。

b. 検索ボックスに、作成したロールの名前 () を入力しますcodecatalyst-build-role。

c. リストからロールを選択します。

ロールの概要ページが表示されます。

d. 上部で、ARN 値をコピーします。

これで、適切なアクセス許可を持つビルドロールを作成し、その ARN を取得しました。

スタックロールを作成するには

> **Note**
>
> セキュリティ上の理由から、スタックロールを作成する必要はありませんが、作成することをお勧めします。スタックロールを作成しない場合は、この手順で詳しく説明するアクセス許可ポリシーをデプロイロールに追加する必要があります。

1. スタックをデプロイするアカウント AWS を使用して にサインインします。
2. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
3. ナビゲーションペインで、ロール を選択します。次に、ロールの作成 を選択します。
4. 上部で、AWS サービス を選択します。
5. サービスのリストから、 を選択しますCloudFormation。
6. [次のステップ: アクセス許可] を選択します。
7. 検索ボックスに、スタック内のリソースにアクセスするために必要なポリシーを追加します。例えば、スタックに AWS Lambda 関数が含まれている場合は、Lambda へのアクセスを許可するポリシーを追加する必要があります。

   > **Tip**
   >
   > 追加するポリシーが不明な場合は、現時点では省略できます。アクションをテストするときに、適切なアクセス許可がない場合、 は追加する必要があるアクセス許可を示すエラー AWS CloudFormation を生成します。

8. [次へ : タグ] を選択します。
9. [次へ: レビュー] を選択します。
10. ロール名 には、次のように入力します。

    ```
    codecatalyst-stack-role
    ```

11. [ロールを作成] を選択します。
12. スタックロールの ARN を取得するには、次の手順を実行します。

    a. ナビゲーションペインで、[ロール] を選択します。

    b. 検索ボックスに、作成したロールの名前 () を入力しますcodecatalyst-stack-role。

c. リストからロールを選択します。

d. 概要ページで、ロール ARN 値をコピーします。

AWS CloudFormation を使用して IAM でポリシーとロールを作成する

AWS CloudFormation テンプレートを作成して使用すると、 CodeCatalyst プロジェクトやワークフロー AWS アカウント の のリソースにアクセスするために必要なポリシーとロールを作成できます。 AWS CloudFormation は AWS 、リソースをモデル化してセットアップするのに役立つサービスです。これにより、リソースの管理に費やす時間を減らし、 で実行されるアプリケーションに集中できます AWS。複数の でロールを作成する場合は AWS アカウント、テンプレートを作成すると、このタスクをより迅速に実行できます。

次のサンプルテンプレートは、デプロイアクションのロールとポリシーを作成します。

```
Parameters:
  CodeCatalystAccountId:
    Type: String
    Description: Account ID from the connections page
  ExternalId:
    Type: String
    Description: External ID from the connections page
Resources:
  CrossAccountRole:
    Type: 'AWS::IAM::Role'
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Principal:
              AWS:
                - !Ref CodeCatalystAccountId
            Action:
              - 'sts:AssumeRole'
            Condition:
              StringEquals:
                sts:ExternalId: !Ref ExternalId
      Path: /
      Policies:
        - PolicyName: CodeCatalyst-CloudFormation-action-policy
          PolicyDocument:
```

```
            Version: "2012-10-17"
            Statement:
              - Effect: Allow
                Action:
                  - 'cloudformation:CreateStack'
                  - 'cloudformation:DeleteStack'
                  - 'cloudformation:Describe*'
                  - 'cloudformation:UpdateStack'
                  - 'cloudformation:CreateChangeSet'
                  - 'cloudformation:DeleteChangeSet'
                  - 'cloudformation:ExecuteChangeSet'
                  - 'cloudformation:SetStackPolicy'
                  - 'cloudformation:ValidateTemplate'
                  - 'cloudformation:List*'
                  - 'iam:PassRole'
                Resource: '*'
```

ウェブアプリケーションのブループリント用のロールの手動作成

CodeCatalyst ウェブアプリケーションのブループリントでは、CDK のビルドロール 、デプロイロール 、スタックロール と呼ばれる、作成した IAM ロールを使用します。

IAM でロールを作成するには、次の手順に従います。

ビルドロールを作成するには

1. 次のように、ロールのポリシーを作成します。

    a. にサインインします AWS。
    b. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
    c. ナビゲーションペインで、ポリシー を選択します。
    d. [Create Policy (ポリシーの作成)] を選択します。
    e. [JSON] タブを選択します。
    f. 既存のコードを削除します。
    g. 次のコードを貼り付けます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": [
                "cloudformation:*",
                "ecr:*",
                "ssm:*",
                "s3:*",
                "iam:PassRole",
                "iam:GetRole",
                "iam:CreateRole",
                "iam:AttachRolePolicy",
                "iam:PutRolePolicy"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> ロールを初めて使用してワークフローアクションを実行するときは、リソースポリシーステートメントでワイルドカードを使用し、使用可能になった後にリソース名でポリシーの範囲を絞り込みます。
>
> ```
> "Resource": "*"
> ```

h. [次へ : タグ] を選択します。

i. [次へ: レビュー] を選択します。

j. 名前 で、次のように入力します。

```
codecatalyst-webapp-build-policy
```

k. [ポリシーの作成] を選択します。

これで、アクセス許可ポリシーが作成されました。

2. 次のようにビルドロールを作成します。

a. ナビゲーションペインで ロール を選択してから、ロールを作成する を選択します。

b. カスタム信頼ポリシー を選択します。

c. 既存のカスタム信頼ポリシーを削除します。

d. 次のカスタム信頼ポリシーを追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

e. [次へ] をクリックします。

f. アクセス許可ポリシーをビルドロールにアタッチします。「アクセス許可の追加」ページの「アクセス許可ポリシー」セクションで、 を検索codecatalyst-webapp-build-policyしてそのチェックボックスをオンにします。

g. [次へ] をクリックします。

h. ロール名 には、次のように入力します。

```
codecatalyst-webapp-build-role
```

i. ロールの説明 には、次のように入力します。

```
CodeCatalyst Web app build role
```

j. [ロールを作成] を選択します。

これで、信頼ポリシーとアクセス許可ポリシーを使用してビルドロールが作成されました。

3. 次のように、アクセス許可ポリシーをビルドロールにアタッチします。

a. ナビゲーションペインでロール を選択し、 を検索しますcodecatalyst-webapp-build-role。

b. を選択して詳細codecatalyst-webapp-build-roleを表示します。

c. 「アクセス許可」タブで「アクセス許可の追加」を選択し、「ポリシーのアタッチ」を選択します。

d. を検索しcodecatalyst-webapp-build-policy、そのチェックボックスを選択し、ポリシーのアタッチ を選択します。

これで、アクセス許可ポリシーをビルドロールにアタッチしました。ビルドロールに、アクセス許可ポリシーと信頼ポリシーの 2 つのポリシーが追加されました。

4. 次のように、ビルドロール ARN を取得します。

a. ナビゲーションペインで、[ロール] を選択します。

b. 検索ボックスに、作成したロールの名前 () を入力しますcodecatalyst-webapp-build-role。

c. リストからロールを選択します。

ロールの概要ページが表示されます。

d. 上部で、ARN 値をコピーします。

これで、適切なアクセス許可を持つビルドロールを作成し、その ARN を取得しました。

SAM ブループリント用のロールの手動作成

CodeCatalyst SAM ブループリントでは、 のビルドロールと SAM のデプロイロール CloudFormationと呼ばれる、作成した IAM ロールを使用します。

IAM でロールを作成するには、次の手順に従います。

のビルドロールを作成するには CloudFormation

1. 次のように、ロールのポリシーを作成します。

a. にサインインします AWS。

b. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。

c. ナビゲーションペインで、ポリシー を選択します。

d. [Create Policy (ポリシーの作成)] を選択します。

e. [JSON] タブを選択します。

f. 既存のコードを削除します。

g. 次のコードを貼り付けます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:*",
                "cloudformation:*"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> ロールを初めて使用してワークフローアクションを実行するときは、リソースポリシーステートメントでワイルドカードを使用し、使用可能になった後にリソース名でポリシーの範囲を絞り込みます。
>
> ```
> "Resource": "*"
> ```

h. [次へ : タグ] を選択します。

i. [次へ: レビュー] を選択します。

j. 名前 で、次のように入力します。

```
codecatalyst-SAM-build-policy
```

k. [ポリシーの作成] を選択します。

これで、アクセス許可ポリシーが作成されました。

2. 次のようにビルドロールを作成します。

a. ナビゲーションペインで ロール を選択してから、ロールを作成する を選択します。

b. カスタム信頼ポリシー を選択します。

c. 既存のカスタム信頼ポリシーを削除します。

d. 次のカスタム信頼ポリシーを追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

e. [次へ] をクリックします。

f. アクセス許可ポリシーをビルドロールにアタッチします。「アクセス許可の追加」ページの「アクセス許可ポリシー」セクションで、そのチェックボックスを検索codecatalyst-SAM-build-policyして選択します。

g. [次へ] をクリックします。

h. ロール名 には、次のように入力します。

```
codecatalyst-SAM-build-role
```

i. ロールの説明 には、次のように入力します。

```
CodeCatalyst SAM build role
```

j. [ロールを作成] を選択します。

これで、信頼ポリシーとアクセス許可ポリシーを使用してビルドロールが作成されました。

3. 次のように、アクセス許可ポリシーをビルドロールにアタッチします。

a. ナビゲーションペインでロール を選択し、 を検索しますcodecatalyst-SAM-build-role。

b. を選択して詳細codecatalyst-SAM-build-roleを表示します。

c. 「アクセス許可」タブで「アクセス許可の追加」を選択し、「ポリシーのアタッチ」を選択します。

d. を検索しcodecatalyst-SAM-build-policy、そのチェックボックスを選択し、ポリシーのアタッチ を選択します。

これで、アクセス許可ポリシーをビルドロールにアタッチしました。ビルドロールに、アクセス許可ポリシーと信頼ポリシーの 2 つのポリシーが追加されました。

4. 次のように、ビルドロール ARN を取得します。

a. ナビゲーションペインで、[ロール] を選択します。

b. 検索ボックスに、作成したロールの名前 () を入力しますcodecatalyst-SAM-build-role。

c. リストからロールを選択します。

ロールの概要ページが表示されます。

d. 上部で、ARN 値をコピーします。

これで、適切なアクセス許可を持つビルドロールを作成し、その ARN を取得しました。

SAM のデプロイロールを作成するには

1. 次のように、ロールのポリシーを作成します。

a. にサインインします AWS。

b. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。

c. ナビゲーションペインで、ポリシー を選択します。

d. [Create Policy (ポリシーの作成)] を選択します。

e. [JSON] タブを選択します。

f. 既存のコードを削除します。

g. 次のコードを貼り付けます。

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "iam:PassRole",
                "iam:DeleteRole",
                "iam:GetRole",
                "iam:TagRole",
                "iam:CreateRole",
                "iam:AttachRolePolicy",
                "iam:DetachRolePolicy",
                "cloudformation:*",
                "lambda:*",
                "apigateway:*"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> ロールを初めて使用してワークフローアクションを実行するときは、リソースポリシーステートメントでワイルドカードを使用し、使用可能になった後にリソース名でポリシーの範囲を絞り込みます。
>
> ```
> "Resource": "*"
> ```

h. [次へ : タグ] を選択します。

i. [次へ: レビュー] を選択します。

j. 名前 で、次のように入力します。

```
codecatalyst-SAM-deploy-policy
```

k. [ポリシーの作成] を選択します。

これで、アクセス許可ポリシーが作成されました。

2. 次のようにビルドロールを作成します。

a. ナビゲーションペインで ロール を選択してから、ロールを作成する を選択します。

b. カスタム信頼ポリシー を選択します。

c. 既存のカスタム信頼ポリシーを削除します。

d. 次のカスタム信頼ポリシーを追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                 ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

e. [次へ] をクリックします。

f. アクセス許可ポリシーをビルドロールにアタッチします。「アクセス許可の追加」ページの「アクセス許可ポリシー」セクションで、そのチェックボックスを検索codecatalyst-SAM-deploy-policyして選択します。

g. [次へ] をクリックします。

h. ロール名 には、次のように入力します。

```
codecatalyst-SAM-deploy-role
```

i. ロールの説明 には、次のように入力します。

```
CodeCatalyst SAM deploy role
```

j. [ロールを作成] を選択します。

これで、信頼ポリシーとアクセス許可ポリシーを使用してビルドロールが作成されました。

3. 次のように、アクセス許可ポリシーをビルドロールにアタッチします。

   a. ナビゲーションペインでロール を選択し、 を検索しますcodecatalyst-SAM-deploy-role。

   b. を選択して詳細codecatalyst-SAM-deploy-roleを表示します。

   c. アクセス許可タブで、アクセス許可の追加 を選択し、ポリシーのアタッチ を選択します。

   d. を検索しcodecatalyst-SAM-deploy-policy、そのチェックボックスをオンにして、ポリシーのアタッチ を選択します。

   これで、アクセス許可ポリシーをビルドロールにアタッチしました。ビルドロールに、アクセス許可ポリシーと信頼ポリシーの 2 つのポリシーが追加されました。

4. 次のように、ビルドロール ARN を取得します。

   a. ナビゲーションペインで、[ロール] を選択します。

   b. 検索ボックスに、作成したロールの名前 () を入力しますcodecatalyst-SAM-deploy-role。

   c. リストからロールを選択します。

   ロールの概要ページが表示されます。

   d. 上部で、ARN 値をコピーします。

   これで、適切なアクセス許可を持つビルドロールを作成し、その ARN を取得しました。

Amazon のコンプライアンス検証 CodeCatalyst

AWS のサービス が特定のコンプライアンスプログラムの範囲内にあるかどうかを確認するには、コンプライアンスプログラムAWS のサービス による対象範囲内のコンプライアンスプログラムを参照し、関心のあるコンプライアンスプログラムを選択します。一般的な情報については、AWS「コンプライアンスプログラム」を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「 でのレポートのダウンロード AWS Artifact」の」を参照してください。

を使用する際のお客様のコンプライアンス責任 AWS のサービス は、お客様のデータの機密性、貴社のコンプライアンス目的、適用される法律および規制によって決まります。 では、コンプライアンスに役立つ以下のリソース AWS を提供しています。

- セキュリティとコンプライアンスのクイックスタートガイド – これらのデプロイガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスに重点を置いたベースライン環境 AWS を にデプロイする手順について説明します。
- アマゾン ウェブ サービスにおける HIPAA セキュリティとコンプライアンスのアーキテクチャ — このホワイトペーパーでは、企業が AWS を使用して HIPAA 対象アプリケーションを作成する方法について説明します。

> **Note**
>
> すべて AWS のサービス HIPAA の対象となるわけではありません。詳細については、「HIPAA 対応サービスのリファレンス」を参照してください。

- AWS コンプライアンスリソース – このワークブックとガイドのコレクションは、お客様の業界や地域に適用される場合があります。
- AWS カスタマーコンプライアンスガイド – コンプライアンスの観点から責任共有モデルを理解します。このガイドでは、ガイダンスを保護し AWS のサービス 、複数のフレームワーク (米国国立標準技術研究所 (NIST)、Payment Card Industry Security Standards Council (PCI)、国際標準化機構 (ISO) を含む) のセキュリティコントロールにマッピングするためのベストプラクティスをまとめています。
- 「 デベロッパーガイド」の「 ルールによるリソースの評価」 – この AWS Config サービスは、リソース設定が社内プラクティス、業界ガイドライン、および規制にどの程度準拠しているかを評価します。 AWS Config
- AWS Security Hub – これにより AWS のサービス 、 内のセキュリティ状態を包括的に把握できます AWS。Security Hub では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールのリストについては、「Security Hub のコントロールリファレンス」を参照してください。
- Amazon GuardDuty – これにより AWS アカウント、疑わしいアクティビティや悪意のあるアクティビティがないか環境を監視することで、、、ワークロード、コンテナ、データに対する潜在的な脅威 AWS のサービス を検出します。 GuardDuty は、特定のコンプライアンスフレームワークで義務付けられている侵入検知要件を満たすことで、PCI DSS などのさまざまなコンプライアンス要件への対応に役立ちます。
- AWS Audit Manager – これにより AWS のサービス 、 AWS 使用状況を継続的に監査し、リスクの管理方法と規制や業界標準への準拠を簡素化できます。

Amazon レジリエンス CodeCatalyst

AWS のグローバルインフラストラクチャは AWS リージョン とアベイラビリティーゾーンを中心として構築されます。リージョンには、低レイテンシー、高いスループット、そして高度の冗長ネットワークで接続されている複数の物理的に独立および隔離されたアベイラビリティーゾーンがあります。アベイラビリティーゾーンでは、ゾーン間で中断することなく自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性が高く、フォールトトレラントで、スケーラブルです。

AWS リージョン とアベイラビリティーゾーンの詳細については、「AWS グローバルインフラストラクチャ」を参照してください。 CodeCatalyst どのデータがレプリケートされるかについての詳細はAWS リージョン、「」を参照してください。Amazon データ保護 CodeCatalyst

Amazon のインフラストラクチャセキュリティ CodeCatalyst

マネージド型サービスとして、Amazon CodeCatalyst AWS はグローバルネットワークセキュリティによって保護されています。AWS セキュリティサービスおよび、AWS がインフラストラクチャを保護する方法については、「AWS クラウドセキュリティ」を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して AWS 環境を設計するには、「セキュリティの柱 AWS 適切なアーキテクチャを備えたフレームワーク」内の「インフラストラクチャ保護」を参照してください。

AWS公開されている API CodeCatalyst 呼び出しを使用してネットワーク経由でアクセスします。クライアントは以下をサポートする必要があります。

- Transport Layer Security (TLS) TLS 1.2 および TLS 1.3 をお勧めします。
- DHE (Ephemeral Diffie-Hellman) や ECDHE (Elliptic Curve Ephemeral Diffie-Hellman) などの Perfect Forward Secrecy (PFS) を使用した暗号スイートです。これらのモードは、Java 7 以降など、最近のほとんどのシステムでサポートされています。

また、リクエストは、アクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token Service (AWS STS) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

Amazon での設定と脆弱性の分析 CodeCatalyst

構成および IT 管理は、AWS とお客様の間で共有される責任です。詳細については、AWS 責任共有モデルを参照してください。

Amazon におけるお客様のデータとプライバシー CodeCatalyst

Amazon CodeCatalyst はお客様のプライバシーを真剣に受け止めており、お客様の情報のセキュリティは最優先事項です。当社がお客様の情報をどのように取り扱うかについての詳細は、AWSプライバシーに関する通知をご覧ください。

データをリクエストして表示するには、の「データのリクエスト」を参照してください。AWS 全般のリファレンス

AWSBuilder ID プロファイルの削除

プロフィールの削除は永久的な操作であり、元に戻すことはできません。削除処理は、[削除] を選択した直後に開始されます。Amazon CodeCatalyst は、お客様のプロフィールと関連するすべての個人情報の削除を開始します。この処理が完了するまでに最大 90 日かかる場合があります。

プロフィールを削除すると、Amazon 内のデータにアクセスしたり、データを復元したりできなくなります CodeCatalyst。これには、個人アクセストークン、ロール、ユーザーメンバーシップ、 CodeCatalystおよびあなたが唯一のメンバーであるすべてのAmazonスペースが含まれます。Amazon にサインインできなくなりました CodeCatalyst。

AWSBuilder ID プロファイルを削除する方法については、の「AWSBuilder ID の削除」を参照してくださいAWS 全般のリファレンス。

Amazon のワークフローアクションのベストプラクティス CodeCatalyst

でワークフローを開発する際に考慮すべきセキュリティのベストプラクティスは多数あります CodeCatalyst。以下は一般的なガイドラインであり、完全なセキュリティソリューションではありません。これらのベストプラクティスは顧客の環境に必ずしも適切または十分でない可能性があるので、処方箋ではなく、あくまで有用な検討事項とお考えください。

トピック

- 機密情報
- ライセンス条項
- 信頼できないコード
- GitHub アクション

機密情報

YAML には機密情報を埋め込まないでください。認証情報、キー、またはトークンをYAMLに埋め込むのではなく、シークレットを使用することをおすすめします。 CodeCatalyst シークレットを使用すると、YAML 内から機密情報を簡単に保存して参照できます。

ライセンス条項

使用することを選択したアクションのライセンス条件に十分注意してください。

信頼できないコード

アクションは通常、プロジェクト、スペース、またはより広いコミュニティで共有できる、自己完結型の単一目的のモジュールです。他社のコードを使用することは、利便性と効率性を大幅に向上させるだけでなく、新たな脅威ベクトルをもたらすことにもなります。以下のセクションを読んで、CI/CD ワークフローを安全に保つためのベストプラクティスに従っていることを確認してください。

GitHub アクション

GitHub アクションはオープンソースで、コミュニティによって構築、管理されています。弊社は責任分担モデルに従い、 GitHub Actions のソースコードをお客様が責任を負う顧客データと見なしています。 GitHub アクションには、シークレット、リポジトリトークン、ソースコード、アカウントリンク、および計算時間へのアクセスを許可できます。 GitHub 実行する予定のアクションの信頼性とセキュリティに自信があることを確認してください。

アクションに関するより具体的なガイダンスとセキュリティのベストプラクティス: GitHub

- セキュリティ強化
- pwn 要求の阻止
- 信頼できない入力
- ビルディングブロックを信頼する方法

CodeCatalyst 信頼モデルを理解する

Amazon CodeCatalyst 信頼モデルにより CodeCatalyst 、 は接続された でサービスロールを引き受けることができます AWS アカウント。このモデルは、IAM ロール、 CodeCatalyst サービスプリンシパル、および CodeCatalyst スペースを接続します。信頼ポリシーは、 `aws:SourceArn`条件キーを使用して、 条件キーで指定された CodeCatalyst スペースにアクセス許可を付与します。この条件キーの詳細については、「IAM ユーザーガイド」の「aws:SourceArn」を参照してください。

信頼ポリシーは JSON ポリシードキュメントです。ここに、ロールを委任できる、信頼するプリンシパルを定義します。ロール信頼ポリシーは、IAMのロールに関連付けられている必須のリソースベースのポリシーです。詳細については、「IAM ユーザーガイド」の「用語と概念」を参照してください。 のサービスプリンシパルの詳細については、 CodeCatalyst「」を参照してくださいのサービスプリンシパル CodeCatalyst。

次の信頼ポリシーでは、 Principal要素にリストされているサービスプリンシパルにリソースベースのポリシーからのアクセス許可が付与され、 Conditionブロックを使用してスコープダウンされたリソースへのアクセスを制限します。

```
"Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
             "Principal": {
                "Service": [
                    "codecatalyst-runner.amazonaws.com",
                    "codecatalyst.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:codecatalyst:::space/spaceId/project/*"
                }
            }
        }
    ]
```

信頼ポリシーでは、 CodeCatalyst サービスプリンシパルには、 CodeCatalyst スペース ID の Amazon リソースネーム (ARN) を含む aws:SourceArn条件キーを通じてアクセス許可が付与されます。ARN は次の形式を使用します。

```
arn:aws:codecatalyst:::space/spaceId/project/*
```

> **Important**
>
> スペース ID は、 などの条件キーでのみ使用しますaws:SourceArn。IAM ポリシーステートメントのスペース ID をリソース ARN として使用しないでください。

ベストプラクティスとして、ポリシーで可能な限りアクセス許可の範囲を絞り込みます。

- aws:SourceArn 条件キーでワイルドカード (*) を使用して、 でスペース内のすべてのプロジェクトを指定できますproject/*。
- を使用して、スペース内の特定のプロジェクトのaws:SourceArn条件キーでリソースレベルのアクセス許可を指定できますproject/*projectId*。

のサービスプリンシパル CodeCatalyst

リソースベースの JSON ポリシーの Principal要素を使用して、リソースへのアクセスを許可または拒否するプリンシパルを指定します。信頼ポリシーで指定できるプリンシパルには、ユーザー、ロール、アカウント、およびサービスが含まれます。アイデンティティベースのポリシーでは Principal要素を使用できません。同様に、グループが認証ではなくアクセス許可と関連し、プリンシパルが認証された IAM エンティティであるため、ポリシー (リソースベースのポリシーなど) でユーザーグループをプリンシパルとして識別することはできません。

信頼ポリシーでは、リソースベースのポリシーの AWS のサービス Principal要素またはプリンシパルをサポートする条件キーで を指定できます。サービスプリンシパルは、サービスによって定義されます。に定義されているサービスプリンシパルを次に示します CodeCatalyst。

- codecatalyst.amazonaws.com - このサービスプリンシパルは、 CodeCatalyst へのアクセスを許可するロールに使用されます AWS。
- codecatalyst-runner.amazonaws.com - このサービスプリンシパルは、 CodeCatalyst ワークフローのデプロイの AWS リソース CodeCatalyst へのアクセスを許可するロールに使用されます。

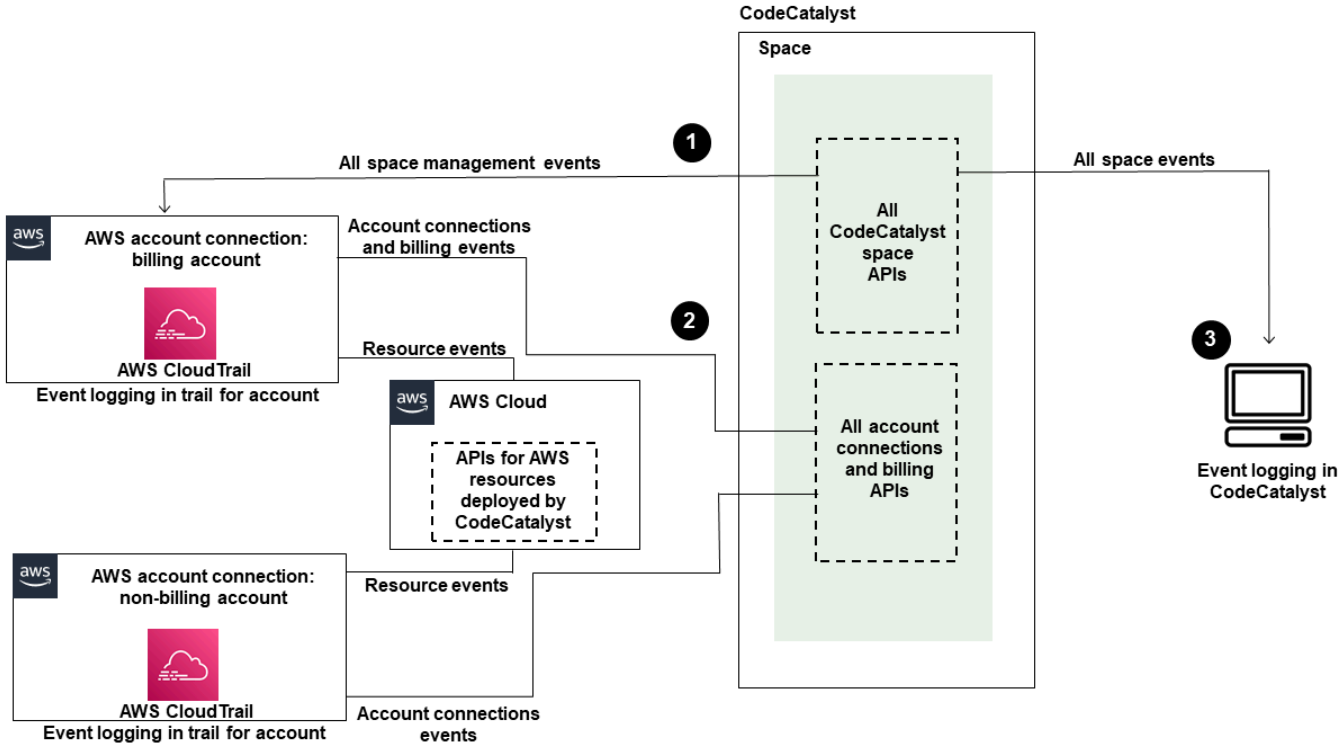
詳細については、「IAM ユーザーガイド」の「AWS JSON ポリシーの要素: プリンシパル」を参照してください。

ログ記録を使用したイベントと API コールのモニタリング

Amazon では CodeCatalyst、スペースの管理イベントは によって収集 AWS CloudTrail され、そのスペースの請求アカウントの証跡に記録されます。 CloudTrail ログ記録は CodeCatalyst 、イベントのログ記録を管理する主な方法です。セカンダリメソッドは、 でイベントログ記録を表示しています CodeCatalyst。

アカウントのイベントは、 用に設定された証跡と指定されたバケットでログに記録されます AWS アカウント。

次の図は、スペースのすべての管理イベントが請求アカウント CloudTrail にログインし、アカウント接続/請求イベントと AWS リソースイベントがそれぞれのアカウント CloudTrail にログインする方法を示しています。



この図表は以下のステップを示しています。

1. スペースが作成されると、 AWS アカウント はスペースに接続され、請求アカウントとして指定されます。使用される証跡は、請求アカウント CloudTrail 用に で作成された証跡で、スペースイベントがログに記録されます。 は CodeCatalyst 、スペースによって、またはスペースに代わって行われた API コールおよび関連イベントを CloudTrail キャプチャし、指定した S3 バケットにログファイルを配信します。請求アカウントが別の AWS アカウントに変わると、スペースイベントはそのアカウントの証跡とバケットに記録されます。によってログに記録される CodeCatalyst 管理イベントの詳細については、 CloudTrail「」を参照してくださいCodeCatalyst の情報 CloudTrail。
2. 請求アカウントを含む、スペースに接続された他のアカウントは、アカウント接続と請求イベントのイベントのサブセットを記録します。そのアカウントにデプロイされた AWS リソースのアカウントイベントを生成する CodeCatalyst ワークフローは、スペースによって、またはスペースに代わって行われた AWS アカウント. CloudTrail captures API コールおよび関連イベントの証跡とバケットにも記録され、 CodeCatalyst指定した S3 バケットにログファイルを配信します。に

よってログに記録される CodeCatalyst 管理イベントの詳細については、 CloudTrail「」を参照してくださいイベントログを使用したログイベントへのアクセス。

3. また、 を使用して list-event-logs コマンドを使用して、スペース内の特定の時間内のスペース内の CodeCatalyst アクションをモニタリングすることもできます AWS CLI。詳細については、「Amazon CodeCatalyst API リファレンスガイド」を参照してください。スペース内のアクションのイベントのリストを呼び出すには、スペース管理者ロールが必要です。 CodeCatalyst 詳細については、「イベントログを使用したログイベントへのアクセス」を参照してください。

> **Note**
>
> `ListEventLogs` は、特定のスペースで過去 30 日間のイベントを保証します。 AWS CloudTrail コンソール CodeCatalyst で の過去 90 日間の管理イベントのリストを表示および取得するには、イベント履歴 を表示するか、証跡を作成して過去 90 日間に延長されたイベントの記録を作成および維持します。詳細については、 CloudTrail「イベント履歴の使用」および CloudTrail「証跡の使用」を参照してください。

> **Note**
>
> AWS CodeCatalyst ワークフローの接続されたアカウントにデプロイされた リソースは、CodeCatalyst スペースの CloudTrail ログ記録の一部として記録されません。例えば、CodeCatalyst リソースにはスペースが含まれ、project. AWS resources には Amazon ECS サービスまたは Lambda 関数が含まれます。リソース AWS アカウント がデプロイされる各に対して CloudTrail ログ記録を個別に設定する必要があります。

以下は、 でのイベントモニタリングに考えられるフローの 1 つです CodeCatalyst。

Mary Major は CodeCatalyst スペースのスペース管理者であり、 に記録されているスペース内のスペースレベルおよびプロジェクトレベルのリソース CodeCatalyst に関する のすべての管理イベントを表示します CloudTrail。にログ記録されるイベントの例CodeCatalyst の情報 CloudTrailについては、「」を参照してください CloudTrail。

開発環境など CodeCatalyst、 で作成されたリソースの場合、Mary はスペースの請求アカウントのイベント履歴を表示し、 のプロジェクトメンバーによって開発環境が作成されたイベントを調査します CodeCatalyst。イベントは、開発環境を作成したユーザーの AWS Builder ID の ID ストア IAM ID タイプと認証情報を提供します。サーバーレスデプロイの Lambda 関数など CodeCatalyst、のワークフローによってデプロイされた AWS ときに で作成されたリソースの場合、 AWS アカウン

ト 所有者はワークフローデプロイアクションの個別の AWS アカウント (に接続されたアカウントでもある CodeCatalyst) に関連付けられた証跡のイベント履歴を表示できます。

さらに調査するために、Mary は の list-event-logs コマンドを使用して、スペース内のすべての CodeCatalyst APIs のイベントを表示することもできます AWS CLI。

トピック

- AWS CloudTrail ログ AWS アカウント 記録を使用した API コールのモニタリング
- イベントログを使用したログイベントへのアクセス

AWS CloudTrail ログ AWS アカウント 記録を使用した API コールのモニタリング

Amazon CodeCatalyst は と統合されています。これは AWS CloudTrail、ユーザー、ロール、またはイベント AWS アカウント として接続された CodeCatalyst で に代わって行われた AWS のサービス. CloudTrail captures API コールによって実行されたアクションを記録するサービスです。証跡を作成する場合は、 の CloudTrail イベントなど、S3 バケットへのイベントの継続的な配信を有効にすることができます CodeCatalyst。証跡を設定しない場合でも、 CloudTrail コンソールのイベント履歴 で最新のイベントを表示できます。

CodeCatalyst では、次のアクションをイベントとして CloudTrail ログファイルに記録できます。

- CodeCatalyst スペースの管理イベントは、スペースの指定された請求アカウント AWS アカウント である に記録されます。詳細については、「CodeCatalyst スペースイベント」を参照してください。

  > **Note**
  >
  > CodeCatalyst スペースのデータイベントには、「」で説明されているように CLI を使用してアクセスできますイベントログを使用したログイベントへのアクセス。

- 接続された で発生する CodeCatalyst ワークフローアクションで使用されるリソースのイベントは、その のイベントとして記録 AWS アカウント されます AWS アカウント。詳細については、「CodeCatalyst アカウント接続と請求イベント」を参照してください。

> **Important**
>
> 複数のアカウントをスペースに関連付けることができますが、 CodeCatalyst スペースとプロジェクトのイベントの CloudTrail ログ記録は請求アカウントにのみ適用されます。

スペース請求アカウントは AWS アカウント 、 無料利用枠を超える CodeCatalyst リソースに対して課金される AWS です。複数のアカウントをスペースに接続できますが、指定された請求アカウントは 1 つのアカウントのみです。スペースの請求アカウントまたは追加の接続アカウントには、Amazon ECS クラスターや S3 バケットなどの AWS リソースとインフラストラクチャを CodeCatalyst ワークフローからデプロイするために使用される IAM ロールを含めることができます。ワークフロー YAML を使用して、デプロイ AWS アカウント した を識別できます。

> **Note**
>
> AWS CodeCatalyst ワークフローの接続されたアカウントにデプロイされた リソースは、CodeCatalyst スペースの CloudTrail ログ記録の一部として記録されません。例えば、CodeCatalystリソースにはスペースや project. AWS resources には Amazon ECS サービスや Lambda 関数が含まれます。 CloudTrail ログ記録は、リソース AWS アカウント がデプロイされる各 に対して個別に設定する必要があります。

CodeCatalyst 接続されたアカウントのログインには、次の考慮事項が含まれます。

- CloudTrail イベントへのアクセスは、 ではなく、接続されたアカウントの IAM で管理されます CodeCatalyst。
- GitHub リポジトリへのリンクなどのサードパーティー接続では、サードパーティーのリソース名が CloudTrail ログに記録されます。

> **Note**
>
> CloudTrail CodeCatalyst イベントの ログ記録はスペースレベルであり、プロジェクトの境界によってイベントを分離しません。

の詳細については CloudTrail、「AWS CloudTrail ユーザーガイド」を参照してください。

> **Note**
>
> このセクションでは、スペースにログインした と、 に接続されている にログイン CodeCatalyst AWS アカウント したすべてのイベントの CloudTrail ログ記録について説明します CodeCatalyst。さらに、 CodeCatalyst スペースに記録されたすべてのイベントを確認するには、 AWS CLI および aws codecatalyst list-event-logs コマンドを使用することもできます。詳細については、「イベントログを使用したログイベントへのアクセス」を参照してください。

CodeCatalyst スペースイベント

スペースレベルおよびプロジェクトレベルのリソースを管理する CodeCatalyst ための のアクションは、スペースの請求アカウントに記録されます。 CodeCatalyst スペースの CloudTrail ログ記録の場合、イベントは次の考慮事項に従ってログに記録されます。

- CloudTrail イベントはスペース全体に適用され、単一のプロジェクトには限定されません。
- AWS アカウント を CodeCatalyst スペースに接続すると、アカウント接続のログ可能なイベントがその に記録されます AWS アカウント。この接続を有効にすると、無効にすることはできません。
- AWS アカウント を CodeCatalyst スペースに接続し、スペースの請求アカウントとして指定すると、イベントはその に記録されます AWS アカウント。この接続を有効にすると、無効にすることはできません。

  スペースレベルおよびプロジェクトレベルのリソースのイベントは、請求アカウントにのみ記録されます。 CloudTrail 送信先アカウントを変更するには、 で請求アカウントを更新します CodeCatalyst。次の毎月の請求サイクルの開始時に、この変更は の新しい請求アカウントに対して有効になります CodeCatalyst。その後、 CloudTrail送信先アカウントが更新されます。

以下は、スペースレベルおよびプロジェクトレベルのリソースを管理するための の CodeCatalystアクション AWS に関連する のイベントの例です。次の APIsは SDK と CLI を通じてリリースされます。イベントは、 CodeCatalyst スペースの請求アカウントとして AWS アカウント 指定された に記録されます。

- `CreateDevEnvironment`
- `CreateProject`
- `DeleteDevEnvironment`

- GetDevEnvironment
- GetProject
- GetSpace
- GetSubscription
- ListDevEnvironments
- ListDevEnvironmentSessions
- ListEventLogs
- ListProjects
- ListSourceRepositories
- StartDevEnvironment
- StartDevEnvironmentSession
- StopDevEnvironment
- StopDevEnvironmentSession
- UpdateDevEnvironment

CodeCatalyst アカウント接続と請求イベント

アカウント接続または請求の でのアクションに関連する の CodeCatalystイベントの例 AWS を次に示します。

- AcceptConnection
- AssociateIAMRoletoConnection
- DeleteConnection
- DissassociateIAMRolefromConnection
- GetBillingAuthorization
- GetConnection
- GetPendingConnection
- ListConnections
- ListIAMRolesforConnection
- PutBillingAuthorization
- RejectConnection

CodeCatalyst の情報 CloudTrail

CloudTrail アカウントを作成する AWS アカウント と、 で が有効になります。これを CodeCatalyst スペースに接続する AWS アカウント と、ログインしている で発生したそのスペースのイベント AWS アカウント はその AWS アカウント CloudTrail にログインします。のログ可能なイベント CodeCatalyst は、接続 CloudTrail されたアカウントの CloudTrail ログと CloudTrail コンソールのイベント履歴に、そのアカウントの他のログ可能な AWS イベントとともに記録されます。

各イベントまたはログエントリには、誰がリクエストを生成したかという情報が含まれます。アイデンティティ情報は、以下を判別するのに役立ちます:

- リクエストが Builder ID AWS を持つユーザーによって行われたかどうか。
- リクエストがルートまたは AWS Identity and Access Management (IAM) ユーザーの認証情報を使用して行われたかどうか。
- リクエストがロールまたはフェデレーションユーザーのテンポラリなセキュリティ認証情報を使用して行われたかどうか。
- リクエストが別の AWS サービスによって行われたかどうか。

詳細については、CloudTrail userIdentity 要素」を参照してください。

CloudTrail イベントへのアクセス

でのアクティビティのイベントなど AWS アカウント、 のイベントの継続的な記録については CodeCatalyst AWS アカウント、証跡を作成します。証跡により、 CloudTrail はログファイルを S3 バケットに配信できます。デフォルトでは、コンソールで証跡を作成するときに、証跡がすべての AWS リージョンに適用されます。証跡は、 AWS パーティション内のすべてのリージョンからのイベントをログに記録し、指定した S3 バケットにログファイルを配信します。さらに、 CloudTrail ログで収集されたイベントデータをさらに分析し、それに基づいて行動するように他の AWS サービスを設定できます。詳細については、次を参照してください:

- 「証跡作成の概要」
- CloudTrail がサポートするサービスと統合
- CloudTrail の Amazon SNS 通知の設定
- 複数のリージョンからの CloudTrail ログファイルの受信と複数のアカウントからの CloudTrail ログファイルの受信

証跡は、指定した S3 バケットにイベントをログファイルとして配信できるようにする設定です。CloudTrail ログファイルには 1 つ以上のログエントリが含まれます。イベントは任意のソースからの単一のリクエストを表し、リクエストされたアクション、アクションの日時、リクエストパラメータなどに関する情報が含まれます。 CloudTrail ログファイルはパブリック API コールの順序付けられたスタックトレースではないため、特定の順序では表示されません。

の CodeCatalyst アカウント接続イベントの例 AWS

次の例は、 `ListConnections`アクションを示す CloudTrail ログエントリを示しています。スペース AWS アカウント に接続されている の場合、 `ListConnections` はこの CodeCatalyst の へのすべてのアカウント接続を表示するために使用されます AWS アカウント。イベントは で AWS アカウント 指定された に記録され`accountId`、 の値はアクションに使用されるロールの Amazon リソースネーム (ARN) `arn`になります。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "role-ARN",
        "accountId": "account-ID",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AKIAI44QH8DHBEXAMPLE",
                "arn": "role-ARN",
                "accountId": "account-ID",
                "userName": "user-name"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2022-09-06T15:04:31Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2022-09-06T15:08:43Z",
    "eventSource": "account-ID",
    "eventName": "ListConnections",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.168.0.1",
```

```
    "userAgent": "aws-cli/1.18.147 Python/2.7.18 Linux/5.4.207-126.363.amzn2int.x86_64
 botocore/1.18.6",
    "requestParameters": null,
    "responseElements": null,
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 ",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 ",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "account-ID",
    "eventCategory": "Management"
}
```

の CodeCatalyst プロジェクトリソースイベントの例 AWS

次の例は、 CreateDevEnvironmentアクションを示す CloudTrail ログエントリを示しています。スペース AWS アカウント に接続され、スペースの指定された請求アカウントである は、開発環境の作成など、スペース内のプロジェクトレベルのイベントに使用されます。

の accountIdフィールドuserIdentityで、これはすべての AWS Builder ID の ID プールをホストする IAM Identity Center アカウント ID (432677196278) です。このアカウント ID には、イベントの CodeCatalyst ユーザーに関する以下の情報が含まれています。

- type フィールドは、リクエストの IAM エンティティのタイプを示します。スペースとプロジェクトリソースの CodeCatalyst イベントの場合、この値は ですIdentityCenterUser。accountId フィールドは、認証情報の取得に使用されたエンティティを所有するアカウントを指定します。
- userId フィールドには、ユーザーの AWS Builder ID 識別子が含まれます。
- identityStoreArn フィールドには、ID ストアアカウントとユーザーのロール ARN が含まれます。

recipientAccountId フィールドには、スペースの請求アカウントのアカウント ID が含まれ、ここでは 111122223333 のサンプル値が含まれます。

詳細については、CloudTrail userIdentity 要素」を参照してください。

```
{
 "eventVersion": "1.09",
 "userIdentity": {
```

```
  "type": "IdentityCenterUser",
  "accountId": "432677196278",
  "onBehalfOf": {
   "userId": "user-ID",
   "identityStoreArn": "arn:aws:identitystore::432677196278:identitystore/d-9067642ac7"
  },
  "credentialId": "ABCDefGhiJKLMn11Lmn_1AbCDEFgHijk-AaBCdEFGHIjKLmnOPqrs11abEXAMPLE"
 },
 "eventTime": "2023-05-18T17:10:50Z",
 "eventSource": "codecatalyst.amazonaws.com",
 "eventName": "CreateDevEnvironment",
 "awsRegion": "us-west-2",
 "sourceIPAddress": "192.168.0.1",
 "userAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:102.0) Gecko/20100101
 Firefox/102.0",
 "requestParameters": {
  "spaceName": "MySpace",
  "projectName": "MyProject",
  "ides": [{
   "runtime": "public.ecr.aws/q6e8p2q0/cloud9-ide-runtime:2.5.1",
   "name": "Cloud9"
  }],
  "instanceType": "dev.standard1.small",
  "inactivityTimeoutMinutes": 15,
  "persistentStorage": {
   "sizeInGiB": 16
  }
 },
 "responseElements": {
  "spaceName": "MySpace",
  "projectName": "MyProject",
  "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 "
 },
 "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
 "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
 "readOnly": false,
 "eventType": "AwsApiCall",
 "managementEvent": true,
 "recipientAccountId": "111122223333",
 "sharedEventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
 "eventCategory": "Management"
}
```

> **Note**
>
> 特定のイベントでは、ユーザーエージェントがわからない場合があります。この場合、 は CloudTrail イベントの Unknown `userAgent` フィールドに の値 CodeCatalyst を指定します。

CodeCatalyst イベント証跡のクエリ

Amazon Athena のクエリテーブルを使用して、 CloudTrail ログのクエリを作成および管理できます。クエリの作成の詳細については、「Amazon Athena ユーザーガイド」の AWS CloudTrail 「ログのクエリ」を参照してください。 Amazon Athena

イベントログを使用したログイベントへのアクセス

ユーザーが Amazon でアクションを実行すると CodeCatalyst、これらのアクションはイベントとして記録されます。を使用して AWS CLI 、指定した時間枠内のスペース内のイベントのログを表示できます。これらのイベントを表示して、アクションの日時、アクションを実行したユーザーの名前、ユーザーがリクエストを行った IP アドレスなど、スペースで実行されたアクションを確認できます。

> **Note**
>
> 接続された請求アカウント CloudTrail に対して、 CodeCatalyst スペースの管理イベントがログインされます。によってログに記録される CodeCatalyst 管理イベントの詳細については、 CloudTrail「」を参照してくださいCodeCatalyst の情報 CloudTrail。

スペースのイベントのログを表示するには、 をインストールして のプロファイル AWS CLI で設定し CodeCatalyst、スペースのスペース管理者ロールを持っている必要があります。詳細については、「AWS CLIとを使用するためのセットアップ CodeCatalyst」および「スペース管理者ロール」を参照してください。

> **Note**
>
> 接続された CodeCatalyst で に代わって発生したイベントのログ記録を表示したり AWS アカウント、接続された請求アカウントのスペースまたはプロジェクトリソースのイベントのログ記録を表示したりするには、 を使用できます AWS CloudTrail。詳細については、

「AWS CloudTrail ログ AWS アカウント 記録を使用した API コールのモニタリング」を参照してください。

1. ターミナルまたはコマンドラインを開き、aws codecatalyst list-event-logsコマンドを実行して、以下を指定します。

   - **--space-name** オプションを含むスペースの名前。
   - RFC 3339 で指定された協定世界時 (UTC) タイムスタンプ形式でイベントの確認を開始する日時。 **--start-time**オプション。
   - RFC 3339 で指定されている協定世界時 (UTC) タイムスタンプ形式で、 **--end-time**オプションを指定してイベントの確認を停止する日時。
   - (オプション) **--max-results**オプションを指定して、1 回のレスポンスで返す結果の最大数。結果の数が指定した数より大きい場合、レスポンスには、次の結果を返すために使用できる nextToken 要素が含まれます。
   - (オプション) **--event-name**オプションを指定して、返される特定のイベントタイプに結果を制限します。

   この例では、期間 *2022-11-30 ExampleCorp*から *2022-12-01* までの という名前のスペースにログに記録されたイベントが返され、レスポンスには最大 *2* つのイベントが返されます。

   ```
   aws codecatalyst list-event-logs --space-name ExampleCorp --start-time 2022-11-30
    --end-time 2022-12-01 --event-name list-event-logs --max-results 2
   ```

2. この時間枠にイベントが発生した場合、コマンドは次のような結果を返します。

   ```
   {
       "nextToken": "EXAMPLE",
       "items": [
           {
               "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
               "eventName": "listEventLogs",
               "eventType": "AwsApiCall",
               "eventCategory": "MANAGEMENT",
               "eventSource": "manage",
               "eventTime": "2022-12-01T22:47:24.605000+00:00",
               "operationType": "READONLY",
               "userIdentity": {
   ```

```
                "userType": "USER",
                "principalId": "a1b2c3d4e5-678fgh90-1a2b-3c4d-e5f6-EXAMPLE11111"
                "userName": "MaryMajor"
            },
            "requestId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
            "requestPayload": {
                "contentType": "application/json",
                "data": "{\"spaceName\":\"ExampleCorp\",\"startTime\":
\"2022-12-01T00:00:00Z\",\"endTime\":\"2022-12-10T00:00:00Z\",\"maxResults\":
\"2\"}"
            },
            "sourceIpAddress": "127.0.0.1",
            "userAgent": "aws-cli/2.9.0 Python/3.9.11 Darwin/21.3.0 exe/x86_64
 prompt/off command/codecatalyst.list-event-logs"
        },
        {
            "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
            "eventName": "createProject",
            "eventType": "AwsApiCall",
            "eventCategory": "MANAGEMENT",
            "eventSource": "manage",
            "eventTime": "2022-12-01T09:15:32.068000+00:00",
            "operationType": "MUTATION",
            "userIdentity": {
                "userType": "USER",
                "principalId": "a1b2c3d4e5-678fgh90-1a2b-3c4d-e5f6-EXAMPLE11111",
                "userName": "MaryMajor"
            },
            "requestId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
            "requestPayload": {
                "contentType": "application/json",
                "data": "{\"spaceName\":\"ExampleCorp\",\"name\":\"MyFirstProject
\",\"displayName\":\"MyFirstProject\"}"
            },
            "responsePayload": {
                "contentType": "application/json",
                "data": "{\"spaceName\":\"ExampleCorp\",\"name\":\"MyFirstProject
\",\"displayName\":\"MyFirstProject\",\"id\":\"a1b2c3d4-5678-90ab-cdef-
EXAMPLE4444\"}"
            },
            "sourceIpAddress": "192.0.2.23",
            "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:102.0)
 Gecko/20100101 Firefox/102.0"
        }
```

```
    ]
}
```

3. --next-token オプションと返されたトークンの値を使用してlist-event-logsコマンドを再度実行し、リクエストに一致するログに記録された次のイベントのセットを取得します。

での ID、アクセス許可、アクセスのクォータ CodeCatalyst

次の表は、Amazon での ID、アクセス許可、アクセスのクォータと制限を示しています CodeCatalyst。Amazon のクォータの詳細については、 CodeCatalyst「」を参照してくださいのクォータ CodeCatalyst。

リソース	情報
のエイリアス CodeCatalyst	3～100 文字の文字を使用できる任意の組み合わせで、文字で始まる必要があります。有効な文字: A～Z、a～z、0～9。エイリアスは、次のことはできません。 • 3 文字未満 • スペースまたは以下の文字を含める: ? ^ * [\ ~ :
ユーザーが 1 日あたりに送信する招待の最大数	500
E メールアドレスに送信される 1 日あたりの招待の最大数	25
ユーザーあたりの個人アクセストークン (PAT) の最大数	100
プロバイダータイプごとの、すべてのスペースにおける各ユーザー ID (CodeCatalystエイリアス) の個人接続の最大数	1
のパスワード CodeCatalyst	8～64 文字の任意の文字の組み合わせ。有効な文字: A～Z、a～z、0～9。パスワードには、次の英数字以外の文字を含めることができま

リソース	情報
	す。(~ ! @ # $ % ^ & * _ - + = ` | \ { } [ ] : ; " ' < > , . ? /)
の PAT 名 CodeCatalyst	1 ~ 100 文字の任意の文字の組み合わせ
プロジェクトメンバーの招待の有効期限が切れるまでの時間	24 時間後に有効期限切れ
スペースメンバーの招待の有効期限が切れるまでの時間	24 時間後に有効期限切れ
E メールアドレス検証の有効期限が切れるまでの時間	送信から 10 分後に期限切れになります

トラブルシューティング

このセクションでは、Amazon CodeCatalyst プロファイルへのアクセス中に発生する可能性がある一般的な問題のトラブルシューティングに役立ちます。

サインアップに関する問題

サインアップ中にいくつかの問題が発生する場合があります。いくつかの解決策があります。

E メールアドレスは既に使用されています

入力した E メールが既に使用中で、それを自分のものとして認識している場合は、既にプロフィールが提供されている可能性があります。この既存の ID でサインインします。既存の E メールを所有していない場合は、未使用の別の E メールにサインアップします。

メールの確認を完了させることができない

検証 E メールを受信していない場合

1. スパムアイテム、迷惑メールアイテム、削除済みアイテムのフォルダを確認してください。

> **Note**
>
> この確認メールは、no-reply@signin.aws または no-reply@login.awsapps.com のアドレスから送信されます。これらの送信者メールアドレスからのメールを受け入れ、迷惑メールやスパムとして処理しないように、メールシステムを設定することをお勧めします。

2. 5 分待ってから、受信トレイを更新します。スパム、迷惑メール、削除された項目フォルダを再度確認します。
3. それでも検証 E メールが表示されない場合は、コードの再送信 を選択します。既にそのページを終了している場合は、Amazon CodeCatalystにサインアップするためのワークフローを再起動します。

パスワードが最小要件を満たしていない

セキュリティ上の理由から、パスワードには大文字と小文字の両方、および数字の 8～20 文字を含める必要があります。

サインインに関する問題

パスワードを忘れてしまいました

「パスワードを忘れてしまいました」の手順を実行します。

パスワードが機能しません。

パスワードを設定または変更するときは、常に次の要件に従う必要があります。

- パスワードでは、大文字と小文字が区別されます。
- パスワードの長さは 8～64 文字で、大文字と小文字の両方、数字、英数字以外の文字を少なくとも 1 つ使用する必要があります。
- 最後の 3 つのパスワードは再利用できません
  。

MFA を有効にできない

MFA を有効にするには、多要素認証 (MFA) でサインインするように AWS Builder ID を設定する の手順に従って 1 つ以上の MFA デバイスをプロファイルに追加します。

MFA デバイスを追加できない

別の MFA デバイスを追加できない場合、登録できる MFA デバイスの制限に達している可能性があります。新しい MFA デバイスを追加する前に、既存の MFA デバイスを削除する必要がある場合があります。

MFA デバイスを削除できない

MFA を無効にする場合は、MFA デバイスの削除 の手順に従って MFA デバイスを削除してください。ただし、MFA を有効にしておきたい場合は、既存の MFA デバイスを削除する前に、別の MFA デバイスを追加する必要があります。別の MFA デバイスの追加の詳細については、「多要素認証用のデバイスを登録する方法」を参照してください。

サインアウトに関する問題

サインアウトする場所が見つからない

ページの右上隅で、「サインアウト」を選択します。

サインアウトしても完全にサインアウトされない

システムはすぐにサインアウトするように設計されていますが、完全にサインアウトするには最大で 1 時間かかる場合があります。

失敗したワークフローに対するロールが存在しないというエラーが表示される

問題： ウェブアプリケーションまたはサーバーレスブループリントからプロジェクトを作成した後、ワークフローは次のエラーで失敗します。

CLIENT_ERROR: ロールが存在しません

解決方法： ワークフローを実行するアクセス許可を持つ IAM ロールを設定し、ワークフロー YAML に IAM ロールを追加した後も、IAM ロールをアカウント接続に追加する必要がある可能性があるた

め、ワークフローは失敗します。「」で説明されているように、スペースのアカウント接続に IAM ロールを追加しますアカウント接続への IAM ロールの追加。

失敗したワークフローのロールエラーが表示される

問題： ウェブアプリケーションまたはサーバーレスブループリントからプロジェクトを作成した後、ワークフローは次のエラーで失敗します。

CLIENT_ERROR: ロールが正しく設定されていないか、存在しない

解決方法： プロジェクトが作成されたスペースは、 AWS アカウント 接続をセットアップする必要があるか、アカウント接続リクエストを完了する必要がある場合があります。スペースに既にアクティブな AWS アカウント 接続がある場合は、ワークフローアクションを実行するアクセス許可を持つ IAM ロールを作成して追加します。「」で説明されているように、IAM ロールをアカウント接続に追加しますアカウント接続への IAM ロールの追加。

解決方法： 接続を指定せずにプロジェクトが作成された場合、アカウント接続をデプロイ環境に関連付ける必要があります。スペースに既にアクティブな AWS アカウント 接続があり、IAM ロールが追加されている場合は、「」で説明されているように、IAM ロールとのアカウント接続をデプロイ環境に追加する必要がありますデプロイ環境にアカウント接続と IAM ロールを追加する。

プロジェクトワークフローで IAM ロールを更新する必要がある

AWS アカウント 接続が完全にセットアップされ、IAM ロールが作成されてアカウント接続に追加されると、プロジェクトワークフローで IAM ロールを更新できます。

1. CI/CD オプションを選択し、ワークフローを選択します。YAML ボタンを選択します。
2. [編集] を選択します。
3. `ActionRoleArn:` フィールドで、IAM ロール ARN を更新された IAM ロール ARN に置き換えます。[検証] を選択します。
4. [Commit] (コミット) を選択します。

   メインラインブランチ上にある場合、ワークフローは自動的に開始されます。それ以外の場合は、ワークフローを再実行するには、 の実行 を選択します。

個人接続を作成した後、 GitHub アカウントのレビューリクエストがある

への個人用接続を作成すると GitHub、 CodeCatalyst アプリが GitHub アプリとして GitHub アカウントにインストールされます。に、更新された読み取りまたは書き込みアクセス許可 CodeCatalyst

を必要とする特定のリソースがある場合、インストールされているアプリのアクセス許可を更新するために GitHub アカウントにアクセスする必要がある場合があります。

1. にサインイン GitHub し、インストールされているアプリのアカウント設定に移動します。プロファイルアイコンを選択し、設定 を選択し、アプリケーション を選択します。
2. インストール済み GitHub アプリケーションタブの、インストール済みアプリケーションのリストで、 にインストールされているアプリケーションを表示します CodeCatalyst。レビューするアクセス許可がある場合は、レビューリクエストリンクが表示されます。
3. リンクを選択し、プロンプトが表示されたら認証情報を確認します。認証情報を入力し、検証を選択します。
4. 新しいアクセス許可を受け入れ、アクセス許可を適用するリポジトリを指定し、保存 を選択します。

サポートフォームへの入力方法

Amazon CodeCatalyst にアクセスするか、サポートフィードバックフォーム に入力します。「リクエスト情報」セクションの「 のサポート方法」に、Amazon のお客様 CodeCatalystであることを記載してください。問題に最大限効率的に対処できるように、できるだけ詳しく説明してください。

で拡張機能を使用してプロジェクトに機能を追加する CodeCatalyst

Amazon CodeCatalyst には、機能の追加や、 以外の製品との統合に役立つ拡張機能が含まれています CodeCatalyst。 CodeCatalyst カタログの拡張機能を使用すると、チームは でのエクスペリエンスをカスタマイズできます CodeCatalyst。

トピック

利用可能なサードパーティーの拡張機能

リソースの統合を選択した拡張機能に応じて、特定の機能を CodeCatalyst プロジェクトに追加できます。

での GitHub リポジトリの統合 CodeCatalyst

GitHub は、デベロッパーがコードを保存および管理できるようにするクラウドベースのサービスです。GitHub リポジトリ拡張機能を使用すると、Amazon CodeCatalyst プロジェクトでリンクされた GitHub リポジトリを使用できます。新しい CodeCatalyst プロジェクトを作成するときに GitHub リポジトリをリンクすることもできます。詳細については、「リンクされたサードパーティーリポジトリを使用したプロジェクトの作成」を参照してください。

> **Note**
>
> - CodeCatalyst プロジェクトでは、空の GitHub リポジトリまたはアーカイブされたリポジトリを使用できません。
> - GitHub リポジトリ拡張機能は GitHub Enterprise Server リポジトリと互換性がありません。

GitHub リポジトリ拡張機能をインストールして設定すると、次のことが可能になります。

- のソース GitHub リポジトリのリストでリポジトリを表示する CodeCatalyst
- ワークフロー定義ファイルを GitHub リポジトリに保存して管理する
- リンクされた GitHub リポジトリに保存されているファイルを作成、読み取り、更新 CodeCatalyst 、開発環境から削除する
- リンクされた GitHub リポジトリのファイルを に保存してインデックスを作成する CodeCatalyst
- 接続された GitHub アカウントの既存のリポジトリを使用して CodeCatalyst プロジェクトを作成する
- 設計図を使用してプロジェクトを作成するとき、または設計図を適用するときに、設計図によって生成されたコードを使用して GitHub リポジトリを作成する
- 開始 CodeCatalyst ワークフローは、リンクされた GitHub リポジトリにコードがプッシュされた場合、またはリンクされた GitHub リポジトリでプルリクエストが作成、変更、または閉じられたときに自動的に実行されます。
- CodeCatalyst ワークフローでリンクされた GitHub リポジトリソースファイルを使用する
- CodeCatalyst ワークフローでの GitHub アクションの読み取りと実行
- CodeCatalyst ワークフロー実行ステータスをリンクされた GitHub リポジトリに送信し、コミットステータスに基づいて GitHub プルリクエストをマージする

での Bitbucket リポジトリの統合 CodeCatalyst

Bitbucket は、デベロッパーがコードを保存および管理できるようにするクラウドベースのサービスです。Bitbucket リポジトリ拡張機能を使用すると、Amazon CodeCatalyst プロジェクトでリンクされた Bitbucket リポジトリを使用できます。新しい CodeCatalyst プロジェクトを作成するときに Bitbucket リポジトリをリンクすることもできます。詳細については、「リンクされたサードパーティーリポジトリを使用したプロジェクトの作成」を参照してください。

> **Note**
>
> - 空の Bitbucket リポジトリまたはアーカイブされた Bitbucket リポジトリを CodeCatalyst プロジェクトで使用することはできません。
> - Bitbucket リポジトリ拡張機能は Bitbucket データセンターリポジトリと互換性がありません。

Bitbucket リポジトリ拡張機能をインストールして設定すると、次のことが可能になります。

- のソースリポジトリのリストで Bitbucket リポジトリを表示する CodeCatalyst
- ワークフロー定義ファイルを Bitbucket リポジトリに保存および管理します。
- リンクされた Bitbucket リポジトリに保存されているファイルを作成、読み取り、更新 CodeCatalyst 、開発環境から削除する
- 接続された Bitbucket アカウントの既存のリポジトリを使用して CodeCatalyst プロジェクトを作成する
- リンクされた Bitbucket リポジトリのファイルを に保存してインデックスを作成する CodeCatalyst
- ブループリントを使用してプロジェクトを作成するとき、またはブループリントを適用するときに、ブループリントによって生成されたコードを使用して Bitbucket リポジトリを作成する
- 開始 CodeCatalyst ワークフローは、リンクされた Bitbucket リポジトリにコードがプッシュされたとき、またはリンクされた Bitbucket リポジトリでプルリクエストが作成、変更、または閉じられたときに自動的に実行されます。
- CodeCatalyst ワークフローでリンクされた Bitbucket リポジトリのソースファイルを使用する
- リンクされた Bitbucket リポジトリに CodeCatalyst ワークフロー実行ステータスを送信し、コミットステータスに基づいて Bitbucket プルリクエストのマージをブロックする

での Jira の問題の統合 CodeCatalyst

Jira は、アジャイル開発チームが作業を計画、割り当て、追跡、報告、管理できるようにするソフトウェアアプリケーションです。Jira ソフトウェア拡張機能を使用すると、Amazon プロジェクトで Jira CodeCatalyst プロジェクトを使用できます。

> **Note**
>
> CodeCatalyst は Jira Software Cloud とのみ互換性があります。

Amazon CodeCatalyst プロジェクトの Jira ソフトウェア拡張機能をインストールして設定すると、次のことが可能になります。

- プロジェクトにリンク CodeCatalyst して から Jira CodeCatalyst プロジェクトにアクセスする
- CodeCatalyst プルリクエストに関する Jira の問題を更新する
- Jira の問題でリンクされた CodeCatalyst プルリクエストのステータスとワークフロー実行を表示する

拡張機能の概念

で拡張機能を使用する際に知っておくべき概念と用語をいくつか紹介します CodeCatalyst。

拡張子

拡張機能は、 CodeCatalyst スペースにインストールしてプロジェクトに新しい機能を追加し、 以外のサービスと統合できるアドオンです CodeCatalyst。拡張機能は、 CodeCatalyst カタログから参照してインストールできます。

CodeCatalyst カタログ

CodeCatalyst カタログは、 で利用可能なすべての拡張機能の一元的なリストです CodeCatalyst。CodeCatalyst カタログを参照して、ソース、ワークフロー CodeCatalyst などの の分野でのチームのエクスペリエンスを向上させる拡張機能を見つけることができます。

接続とリンク

使用または管理するサードパーティーリソースに応じて、 GitHub アカウント、Bitbucket ワークスペース、または Jira プロジェクトを接続する必要があります。次に、 GitHub リポジトリ、Bitbucket リポジトリ、または Jira プロジェクトを CodeCatalyst プロジェクトにリンクする必要があります。

- GitHub リポジトリ： GitHub アカウントを接続し、リポジトリをリンク GitHub します。
- Bitbucket リポジトリ: Bitbucket ワークスペースを接続し、Bitbucket リポジトリをリンクします。
- Jira ソフトウェア: Jira サイトを接続し、Jira プロジェクトをリンクします。

クイックスタート: 拡張機能のインストール、プロバイダーの接続、 でのリソースのリンク CodeCatalyst

このチュートリアルでは、次の 3 つのタスクのチュートリアルを提供します。

1. GitHub リポジトリ 、Bitbucket リポジトリ、または Jira ソフトウェア拡張機能をインストールします。外部サイトでは、サードパーティー CodeCatalyst のリソースに接続して へのアクセスを提供するように求められます。これは、次のステップの一環として行われます。

   > **Important**
   >
   > GitHub リポジトリ 、Bitbucket リポジトリ 、または Jira ソフトウェア拡張機能を CodeCatalyst スペースにインストールするには、スペースにスペース管理者ロールを持つアカウントでサインインする必要があります。

2. GitHub アカウント、Bitbucket ワークスペース、または Jira サイトを に接続します CodeCatalyst。

   > **Important**
   >
   > GitHub アカウント、Bitbucket ワークスペース、または Jira サイトを CodeCatalyst スペースに接続するには、サードパーティーのソースの管理者と CodeCatalyst スペース管理者の両方である必要があります。

> **Important**
>
> リポジトリ拡張をインストールすると、リンク先のリポジトリのコード CodeCatalyst のインデックスが作成され、 に保存されます CodeCatalyst。これにより、コードは で検索できるようになります CodeCatalyst。でリンクされたリポジトリを使用する場合のコードのデータ保護の詳細については CodeCatalyst、「Amazon CodeCatalyst ユーザーガイド」の「データ保護」を参照してください。

> **Note**
>
> GitHub アカウントへの接続を使用している場合は、アイデンティティと CodeCatalyst アイデンティティ間のアイデンティティマッピングを確立するために、個人接続を作成する必要があります GitHub 。詳細については、「個人用接続」および「個人接続による GitHub リソースへのアクセス」を参照してください。

3. GitHub リポジトリ、Bitbucket リポジトリ、または Jira プロジェクトを CodeCatalyst プロジェクトにリンクします。

> **Important**
>
> - GitHub または Bitbucket リポジトリを寄稿者 としてリンクすることはできますが、サードパーティーリポジトリのリンクを解除できるのは、スペース管理者またはプロジェクト管理者 のみです。詳細については、「での GitHub リポジトリ、Bitbucket リポジトリ、Jira プロジェクトのリンク解除 CodeCatalyst」を参照してください。
> - Jira プロジェクトを CodeCatalyst プロジェクトにリンクするには、スペース管理者または CodeCatalyst プロジェクト管理者である必要があります CodeCatalyst 。

> **Note**
>
> - GitHub または Bitbucket リポジトリは、スペース内の 1 つの CodeCatalyst プロジェクトにのみリンクできます。
> - 空のリポジトリ、アーカイブされたリポジトリ GitHub 、または Bitbucket リポジトリを CodeCatalyst プロジェクトで使用することはできません。

- プロジェクト内の CodeCatalyst リポジトリと同じ名前の GitHub または Bitbucket リポジトリをリンクすることはできません。
- GitHub リポジトリ拡張機能は GitHub Enterprise Server リポジトリと互換性がありません。
- Bitbucket リポジトリ拡張機能は Bitbucket データセンターリポジトリと互換性がありません。
- CodeCatalyst プロジェクトは 1 つの Jira プロジェクトにのみリンクできます。Jira プロジェクトは複数のプロジェクトにリンク CodeCatalyst できます。

GitHub リポジトリまたは Bitbucket リポジトリ拡張機能をインストールし、 GitHub アカウントまたは Bitbucket ワークスペースに接続して、新しい CodeCatalyst プロジェクトを作成するときにサードパーティーのリポジトリをリンクすることもできます。詳細については、「リンクされたサードパーティーリポジトリを使用したプロジェクトの作成」を参照してください。

トピック

ステップ 1: CodeCatalyst カタログからサードパーティーの拡張機能をインストールする

で thid-party リソースを使用する最初のステップ CodeCatalyst は、 CodeCatalyst カタログからGitHub リポジトリ拡張機能をインストールすることです。拡張機能をインストールするには、次の手順を実行し、使用するサードパーティーリソースの拡張機能を選択します。GitHub リポジトリと Bitbucket リポジトリを使用すると、 で GitHub または Bitbucket リポジトリを使用できます CodeCatalyst。Jira ソフトウェアを使用すると、 で Jira の問題を管理できます CodeCatalyst。

CodeCatalyst カタログから拡張機能をインストールするには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. CodeCatalyst スペースに移動します。

3. 

   トップメニューの CodeCatalyst CodeCatalyst カタログアイコンを選択して、カタログに移動します。GitHub リポジトリ 、Bitbucket リポジトリ 、または Jira ソフトウェア を検索できます。カテゴリに基づいて拡張をフィルタリングすることもできます。
4. （オプション) 拡張機能のアクセス許可など、拡張機能の詳細を表示するには、拡張機能名を選択します。
5. [Install] (インストール) を選択します。拡張機能に必要なアクセス許可を確認し、続行する場合は、[インストール] を再度選択します。

拡張機能をインストールすると、拡張機能の詳細ページが表示されます。インストールした拡張機能に応じて、接続されたプロバイダーとリンクされたリソースを表示および管理できます。

ステップ 2: サードパーティープロバイダーをスペースに接続する CodeCatalyst

GitHub リポジトリ 、Bitbucket リポジトリ 、または Jira ソフトウェア拡張機能をインストールした後、次のステップは GitHub アカウント、Bitbucket ワークスペース、または Jira サイトを CodeCatalyst スペースに接続することです。

GitHub アカウント、Bitbucket ワークスペース、または Jira サイトを に接続するには CodeCatalyst

- インストールしたサードパーティーの拡張機能に応じて、次のいずれかを実行します。
  - GitHub リポジトリ： GitHub アカウントに接続します。
    1. 「接続された GitHub アカウント」タブで GitHub 「アカウントの接続」を選択して、 の外部サイトに移動します GitHub。
    2. GitHub 認証情報を使用して GitHub アカウントにサインインし、Amazon をインストールするアカウントを選択します CodeCatalyst。

       > **Tip**
       >
       > 以前に GitHub アカウントをスペースに接続したことがある場合は、再承認を求められません。代わりに、複数の GitHub スペースのメンバーまたは共同作業者の場合は拡張機能をインストールする場所を尋ねるダイアログボックスが表示され、1 つの GitHub スペースにのみ属している場合は Amazon CodeCatalyst アプリケーションの設定ページが表示されます。許可するリポジトリアクセス用にア

プリケーションを設定し、 の保存 を選択します。保存ボタンがアクティブでない場合は、設定を変更してから、もう一度試してください。

3. CodeCatalyst が現在および将来のすべてのリポジトリにアクセスできるようにするか、で使用する特定の GitHub リポジトリを選択するかを選択します CodeCatalyst。デフォルトのオプションは、 によってアクセスされる将来の GitHub リポジトリを含め、 GitHub アカウント内のすべてのリポジトリを含めることです CodeCatalyst。
4. に付与されたアクセス許可を確認し CodeCatalyst、インストール を選択します。

GitHub アカウントを に接続すると CodeCatalyst、GitHub リポジトリ拡張の詳細ページが表示されます。このページでは、接続された GitHub アカウントとリンクされた GitHub リポジトリを表示および管理できます。

- Bitbucket リポジトリ : Bitbucket ワークスペースに接続します。

1. 「Connected Bitbucket workspaces」タブで「Connect Bitbucket workspace」を選択して、Bitbucket の外部サイトに移動します。
2. Bitbucket 認証情報を使用して Bitbucket ワークスペースにサインインします。
3. ワークスペースの承認ドロップダウンメニューから、 CodeCatalyst アクセス権を付与する Bitbucket ワークスペースを選択し、アクセス権の付与を選択します。

   a. ワークスペース CodeCatalyst へのアクセス権を初めて付与する場合は、「設定に移動」を選択して Bitbucket の開発モードを有効にします。

   b. ワークスペースの「インストール済みアプリケーション」ページで、「開発モードを有効にする」チェックボックスを選択して のインストールを許可し CodeCatalyst、から再度接続します CodeCatalyst。

   のインストールを許可すると CodeCatalyst、Bitbucket ワークスペース内のすべてのリポジトリにアクセスできます。

> **Tip**
>
> 以前に Bitbucket ワークスペースをスペースに接続したことがある場合、再承認を求めるプロンプトは表示されません。代わりに、複数の Bitbucket ワークスペースのメンバーまたは共同作業者の場合は拡張機能をインストールする場所を尋ねるダイアログが表示され、1 つの Bitbucket ワークスペースにのみ属している場合は Amazon CodeCatalyst アプリケーションの設定ページが表示されます。許

可するワークスペースアクセス用にアプリケーションを設定し、アクセス許可 を選択します。アクセス許可ボタンがアクティブでない場合は、設定を変更してから、もう一度試してください。

Bitbucket ワークスペースを に接続すると CodeCatalyst、Bitbucket リポジトリの拡張の詳細ページが表示されます。このページでは、接続されている Bitbucket ワークスペースとリンクされた Bitbucket リポジトリを表示および管理できます。

- Jira ソフトウェア: Jira サイトを接続します。

  1. 「Connected Jira sites」タブで「Connect Jira site」を選択して、Atlassian Marketplace の外部サイトに移動します。
  2. 今すぐ取得 を選択して、Jira サイト CodeCatalyst へのインストールを開始します。

     > **Note**
     >
     > 以前に Jira サイト CodeCatalyst にインストールしたことがある場合は、通知が送信されます。「開始」を選択して、最後のステップに進みます。

  3. ロールに応じて、次のいずれかを実行します。

     1. Jira サイト管理者の場合は、サイトのドロップダウンメニューから Jira サイトを選択して CodeCatalyst アプリケーションをインストールし、アプリのインストール を選択します。

        > **Note**
        >
        > Jira サイトが 1 つある場合、このステップは表示されず、自動的に次のステップに進みます。

     2. a. Jira 管理者でない場合は、サイトのドロップダウンメニューから Jira サイトを選択してアプリケーションをインストール CodeCatalyst し、アプリケーション のリクエストを選択します。Jira アプリのインストールの詳細については、「アプリをインストールできるユーザー」を参照してください。

        b. CodeCatalyst 入力テキストフィールドにインストールする必要がある理由を入力するか、デフォルトのテキストのままにして、リクエストの送信を選択します。

  4. アプリケーションのインストール CodeCatalyst 時に によって実行されたアクションを確認し、今すぐ取得 を選択します。

5. アプリケーションをインストールしたら、戻る CodeCatalyst を選択して に戻ります CodeCatalyst。

Jira サイトを に接続すると CodeCatalyst、Jira ソフトウェア拡張機能の詳細ページの Connected Jira sites タブで接続されたサイトを表示できます。

ステップ 3: サードパーティーリソースを CodeCatalyst プロジェクトにリンクする

GitHub または Bitbucket リポジトリを使用するか、 で Jira の問題を管理するための 3 番目と最後のステップ CodeCatalyst は、それらを使用する CodeCatalyst プロジェクトにリンクすることです。

拡張の詳細ページから GitHub リポジトリ、Bitbucket リポジトリ、または Jira プロジェクトを CodeCatalyst プロジェクトにリンクするには

- インストールしたサードパーティーの拡張機能と接続したプロバイダーに応じて、次のいずれかを実行します。
  - GitHub リポジトリ: GitHub リポジトリをリンクします。
    1. リンクされた GitHub リポジトリ タブで、 GitHub リポジトリ のリンク を選択します。
    2. GitHub アカウントドロップダウンから、リンクするリポジトリを含む GitHub アカウントを選択します。
    3. GitHub リポジトリドロップダウンから、 CodeCatalyst プロジェクトにリンクするリポジトリを選択します。

       > **Tip**
       > リポジトリの名前がグレー表示になっている場合、そのリポジトリはスペース内の別のプロジェクトに既にリンクされているため、リンクできません。

    4. (オプション) GitHub リポジトリのリストにリポジトリが表示されない場合は、 の Amazon CodeCatalyst アプリケーションでリポジトリアクセスが設定されていない可能性があります GitHub。接続されたアカウントの CodeCatalyst で使用できる GitHub リポジトリを設定できます。

       a. GitHub アカウントに移動し、設定 を選択し、アプリケーション を選択します。

b. Installed GitHub Apps タブで、Amazon CodeCatalyst アプリケーションの設定を選択します。

c. でリンクする GitHub リポジトリへのアクセスを設定するには、次のいずれかを実行します CodeCatalyst。

- 現在および将来のすべてのリポジトリへのアクセスを提供するには、すべてのリポジトリ を選択します。
- 特定のリポジトリへのアクセスを許可するには、リポジトリの選択のみ を選択し、リポジトリの選択 ドロップダウンを選択し、 でリンクを許可するリポジトリを選択します CodeCatalyst。

5. CodeCatalyst プロジェクトドロップダウンメニューから、 GitHub リポジトリをリンクする CodeCatalyst プロジェクトを選択します。

6. [Link (リンク)] を選択します。

で GitHub リポジトリを使用しない場合は CodeCatalyst、プロジェクトから CodeCatalyst リポジトリのリンクを解除できます。リポジトリのリンクが解除されると、そのリポジトリのイベントはワークフロー実行を開始せず、 CodeCatalyst 開発環境でそのリポジトリを使用することはできません。詳細については、「での GitHub リポジトリ、Bitbucket リポジトリ、Jira プロジェクトのリンク解除 CodeCatalyst」を参照してください。

- Bitbucket リポジトリ: Bitbucket リポジトリをリンクします。

1. Linked Bitbucket リポジトリタブで、Link Bitbucket リポジトリ を選択します。

2. Bitbucket ワークスペースドロップダウンから、リンクするリポジトリを含む Bitbucket ワークスペースを選択します。

3. Bitbucket リポジトリドロップダウンから、 CodeCatalyst プロジェクトにリンクするリポジトリを選択します。

> **Tip**
>
> リポジトリの名前がグレー表示になっている場合、そのリポジトリはスペース内の別のプロジェクトに既にリンクされているため、リンクできません。

4. CodeCatalyst プロジェクトのドロップダウンメニューから、Bitbucket リポジトリをリンクする CodeCatalyst プロジェクトを選択します。

5. [Link (リンク)] を選択します。

で Bitbucket リポジトリを使用しなくなった場合は CodeCatalyst、プロジェクトから CodeCatalyst リンクを解除できます。リポジトリのリンクが解除されると、そのリポジトリのイベントはワークフロー実行を開始せず、 CodeCatalyst 開発環境でそのリポジトリを使用することはできません。詳細については、「での GitHub リポジトリ、Bitbucket リポジトリ、Jira プロジェクトのリンク解除 CodeCatalyst」を参照してください。

- Jira ソフトウェア: Jira プロジェクトをリンクします。

  1. Linked Jira projects タブで、Link Jira project を選択します。
  2. Jira サイトのドロップダウンメニューから、リンクするプロジェクトを含む Jira サイトを選択します。
  3. Jira プロジェクトのドロップダウンメニューから、プロジェクトにリンクする CodeCatalyst プロジェクトを選択します。
  4. CodeCatalyst プロジェクトのドロップダウンメニューから、Jira CodeCatalyst プロジェクトにリンクするプロジェクトを選択します。
  5. [Link (リンク)] を選択します。

  Jira プロジェクトが CodeCatalyst プロジェクトにリンクされると、 CodeCatalyst 問題へのアクセスは完全に無効になり、 CodeCatalyst ナビゲーションペインの問題は Jira プロジェクトにリンクする Jira 発行項目に置き換えられます。

  で Jira プロジェクトを使用しない場合は CodeCatalyst、プロジェクトからリンクを解除できます CodeCatalyst。Jira プロジェクトがリンク解除されると、Jira の問題はプロジェクトで CodeCatalyst利用できなくなり、 CodeCatalyst 問題は再び問題プロバイダーになります。詳細については、「での GitHub リポジトリ、Bitbucket リポジトリ、Jira プロジェクトのリンク解除 CodeCatalyst」を参照してください。

GitHub または Bitbucket リポジトリを、コード のソースリポジトリからプロジェクトにリンクすることもできます。詳細については、「接続されたサードパーティープロバイダーからのリソースのリンク」を参照してください。

次のステップ

GitHub リポジトリまたは Bitbucket リポジトリ拡張機能をインストールし、リソースプロバイダーを接続し、サードパーティーのリポジトリを CodeCatalyst プロジェクトにリンクしたら、CodeCatalyst ワークフローや開発環境で使用できます。ブループリントから生成されたコードを使

用して、接続された GitHub アカウントまたは Bitbucket ワークスペースにサードパーティーのリポジトリを作成することもできます。詳細については、「サードパーティーのリポジトリイベント後にワークフローを自動的に開始する」および「開発環境の作成」を参照してください。

Jira ソフトウェア拡張機能をインストールし、Jira サイトを接続し、Jira プロジェクトを CodeCatalyst プロジェクトにリンクし、プルリクエストをリンクすると、 からの更新 CodeCatalyst が Jira プロジェクトに反映されます。プルリクエストを Jira の問題にリンクする方法の詳細については、「」を参照してくださいJira の問題の CodeCatalyst プルリクエストへのリンク。Jira での CodeCatalyst イベントの表示の詳細については、「」を参照してくださいJira の問題での CodeCatalyst イベントの表示。

スペースへの拡張機能のインストール

スペースの拡張機能をインストールして、その CodeCatalyst スペースのプロジェクトに機能を追加できます。 CodeCatalyst カタログアイコン を選択すると、カタログを表示できます 。

拡張機能とその機能の詳細については、「」を参照してください利用可能なサードパーティーの拡張機能。

> **Important**
>
> 拡張機能をインストールするには、スペースにスペース管理者ロールを持つアカウントでサインインする必要があります。

> **Important**
>
> リポジトリ拡張をインストールすると、リンク先のリポジトリのコード CodeCatalyst のインデックスが作成され、 に保存されます CodeCatalyst。これにより、コードは で検索できるようになります CodeCatalyst。でリンクされたリポジトリを使用する場合のコードのデータ保護の詳細については CodeCatalyst、「Amazon CodeCatalyst ユーザーガイド」の「データ保護」を参照してください。

CodeCatalyst カタログから拡張機能をインストールするには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。

2. CodeCatalyst スペースに移動します。
3. 

   トップメニューの CodeCatalyst カタログアイコンを選択して、カタログに移動します。拡張を検索したり、カテゴリに基づいて拡張をフィルタリングしたりできます。
4. (オプション) 拡張機能の名前を選択すると、拡張機能のアクセス許可など、拡張機能の詳細が表示されます。
5. [Install] (インストール) を選択します。拡張機能に必要なアクセス許可を確認し、続行する場合は、[インストール] を再度選択します。

拡張機能をインストールすると、インストールされた拡張機能の詳細ページが表示されます。拡張機能の詳細については、タブを参照してください。詳細ページでは、必要に応じて拡張機能をさらに設定することもできます。

スペースに拡張機能をアンインストールする

以前に CodeCatalyst スペースにインストールされた拡張機能はアンインストールできます。拡張機能をアンインストールすると、その拡張機能に関連するリソースが CodeCatalyst スペースまたはプロジェクトから削除される可能性があります。

> **Important**
>
> 拡張機能をアンインストールするには、スペースにスペース管理者ロールを持つアカウントでサインインする必要があります。

拡張機能を CodeCatalyst スペースからアンインストールするには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. CodeCatalyst スペースに移動します。
3. 次のいずれかを実行して、スペースにインストールされている拡張機能のリストを表示します。

   a. 設定 を選択し、インストールされた拡張機能 を選択します。

   b. トップメ

      ニュー 

      タログアイコンを選択します。

4. アンインストールする拡張機能の設定を選択します。
5. 拡張機能の詳細ページでアンインストールを選択します。
6. 拡張機能のアンインストールダイアログボックスの情報を確認します。手順に従って、アンインストール を選択して拡張機能をアンインストールします。

GitHub アカウント、Bitbucket ワークスペース、Jira サイトの接続 CodeCatalyst

GitHub または Bitbucket リポジトリを使用するか、 で Jira プロジェクトを管理するには CodeCatalyst、まずサードパーティーのソースを CodeCatalyst スペースに接続する必要があります。拡張機能とその機能の詳細については、「」を参照してください利用可能なサードパーティーの拡張機能。

> **Important**
>
> GitHub アカウント、Bitbucket ワークスペース、または Jira サイトを CodeCatalyst スペースに接続するには、サードパーティーのソースの管理者と CodeCatalyst スペース管理者の両方である必要があります。

> **Note**
>
> GitHub アカウントへの接続を使用している場合は、アイデンティティと CodeCatalyst アイデンティティ間のアイデンティティマッピングを確立するために、個人接続を作成する必要があります GitHub 。詳細については、「個人用接続」および「個人接続による GitHub リソースへのアクセス」を参照してください。

GitHub アカウント、Bitbucket ワークスペース、または Jira サイトを に接続するには CodeCatalyst

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. CodeCatalyst スペースに移動します。
3. 次のいずれかを実行して、スペースにインストールされている拡張機能のリストを表示します。

   a. 設定 を選択し、インストールされた拡張機能 を選択します。

b. トップメニュー 
タログアイコンを選択します。

4. 設定する拡張の 1 つとして、リポジトリ 、Bitbucket リポジトリ 、または Jira ソフトウェア を選択します。 GitHub

5. 設定を選択したサードパーティーの拡張機能に応じて、次のいずれかを実行します。

- GitHub リポジトリ： GitHub アカウントに接続します。

  1. 「接続された GitHub アカウント」タブで GitHub 「アカウントの接続」を選択して、 の外部サイトに移動します GitHub。
  2. GitHub 認証情報を使用して GitHub アカウントにサインインし、Amazon をインストールするアカウントを選択します CodeCatalyst。

  > **Tip**
  >
  > 以前に GitHub アカウントをスペースに接続したことがある場合、再承認を求めるプロンプトは表示されません。代わりに、複数の GitHub スペースのメンバーまたは共同作業者の場合は拡張機能をインストールする場所を尋ねるダイアログボックスが表示され、1 つの GitHub スペースにのみ属している場合は Amazon CodeCatalyst アプリケーションの設定ページが表示されます。許可するリポジトリアクセス用にアプリケーションを設定し、保存 を選択します。保存ボタンがアクティブでない場合は、設定を変更してから再試行してください。

  3. CodeCatalyst が現在および将来のすべてのリポジトリにアクセスできるようにするか、で使用する特定の GitHub リポジトリを選択するかを選択します CodeCatalyst。デフォルトのオプションは、 によってアクセスされる将来の GitHub リポジトリを含め、 GitHub アカウント内のすべてのリポジトリを含めることです CodeCatalyst。
  4. に付与されたアクセス許可を確認し CodeCatalyst、インストール を選択します。

  GitHub アカウントを に接続すると CodeCatalyst、GitHub リポジトリ拡張の詳細ページが表示されます。このページでは、接続された GitHub アカウントとリンクされた GitHub リポジトリを表示および管理できます。

- Bitbucket リポジトリ : Bitbucket ワークスペースに接続します。

1. 「Connected Bitbucket workspaces」タブで「Connect Bitbucket workspace」を選択して、Bitbucket の外部サイトに移動します。
2. Bitbucket 認証情報を使用して Bitbucket ワークスペースにサインインします。
3. ワークスペースの承認ドロップダウンメニューから、 CodeCatalyst アクセス権を付与する Bitbucket ワークスペースを選択し、アクセス権の付与を選択します。

   a. ワークスペース CodeCatalyst へのアクセス権を初めて付与する場合は、設定に移動を選択して Bitbucket の開発モードを有効にします。

   b. ワークスペースの「インストール済みアプリケーション」ページで、「開発モードを有効にする」チェックボックスを選択して のインストールを許可し CodeCatalyst、から再度接続します CodeCatalyst。

      のインストールを許可すると CodeCatalyst、Bitbucket ワークスペース内のすべてのリポジトリにアクセスできます。

> **Tip**
>
> 以前に Bitbucket ワークスペースをスペースに接続したことがある場合、再承認を求めるプロンプトは表示されません。代わりに、複数の Bitbucket ワークスペースのメンバーまたは共同作業者の場合は拡張機能をインストールする場所を尋ねるダイアログが表示され、1 つの Bitbucket ワークスペースにのみ属している場合は Amazon CodeCatalyst アプリケーションの設定ページが表示されます。許可するワークスペースアクセス用にアプリケーションを設定し、アクセス許可 を選択します。アクセス許可ボタンがアクティブでない場合は、設定を変更してから、もう一度試してください。

Bitbucket ワークスペースを に接続すると CodeCatalyst、Bitbucket リポジトリの拡張の詳細ページが表示されます。このページでは、接続されている Bitbucket ワークスペースとリンクされた Bitbucket リポジトリを表示および管理できます。

- Jira ソフトウェア: Jira サイトを接続します。

  1. 「Connected Jira sites」タブで「Connect Jira site」を選択して、Atlassian Marketplace の外部サイトに移動します。
  2. 今すぐ取得を選択して、Jira サイト CodeCatalyst へのインストールを開始します。

> **Note**
>
> 以前に Jira サイト CodeCatalyst にインストールしたことがある場合は、通知が送信されます。「開始」を選択して、最後のステップに進みます。

3. ロールに応じて、次のいずれかを実行します。

   1. Jira サイト管理者の場合は、サイトのドロップダウンメニューから Jira サイトを選択して CodeCatalyst アプリケーションをインストールし、アプリのインストール を選択します。

      > **Note**
      >
      > Jira サイトが 1 つある場合、このステップは表示されず、自動的に次のステップに進みます。

   2. a. Jira 管理者でない場合は、サイトのドロップダウンメニューから Jira サイトを選択してアプリケーションをインストール CodeCatalyst し、アプリケーション のリクエストを選択します。Jira アプリのインストールの詳細については、「アプリをインストールできるユーザー」を参照してください。

      b. CodeCatalyst 入力テキストフィールドにインストールする必要がある理由を入力するか、デフォルトのテキストのままにして、リクエストの送信を選択します。

4. アプリケーションのインストール CodeCatalyst 時に によって実行されたアクションを確認し、今すぐ取得 を選択します。

5. アプリケーションをインストールしたら、戻る CodeCatalyst を選択して に戻ります CodeCatalyst。

Jira サイトを に接続すると CodeCatalyst、Jira ソフトウェア拡張の詳細ページの Connected Jira sites タブで接続されたサイトを表示できます。

GitHub でリポジトリ、Bitbucket リポジトリ、または Jira の問題を使用しなくなった場合は CodeCatalyst、サードパーティーのソースを切断できます。 GitHub アカウントまたは Bitbucket ワークスペースが切断されると、サードパーティーリポジトリのイベントはワークフロー実行を開始せず、 CodeCatalyst 開発環境でそれらのリポジトリを使用することはできません。Jira サイトが切断されると、サイトのプロジェクトからの Jira の問題は CodeCatalyst プロジェクトで利用できなく

なり、 CodeCatalyst 問題が再び問題プロバイダーになります。詳細については、「GitHub アカウント、Bitbucket ワークスペース、Jira サイトの切断 CodeCatalyst」を参照してください。

GitHub アカウント、Bitbucket ワークスペース、Jira サイトの切断 CodeCatalyst

で GitHub リポジトリ、Bitbucket リポジトリ、または Jira の問題を使用しない場合は CodeCatalyst、サードパーティーのソースを切断できます。 GitHub アカウントまたは Bitbucket ワークスペースが切断されると、リポジトリ内のイベントはワークフロー実行を開始せず、CodeCatalyst 開発環境でそれらのリポジトリを使用することはできません。Jira サイトが切断されると、サイトのプロジェクトからの Jira の問題は CodeCatalyst プロジェクトで利用できなくなり、CodeCatalyst 問題が再び問題プロバイダーになります。

> **Note**
>
> - GitHub アカウントを切断するには、まずそのアカウントからリンクされたすべての GitHub リポジトリのリンクを解除する必要があります。
> - Bitbucket ワークスペースを切断するには、まず、リンクされたすべての Bitbucket リポジトリをそのワークスペースからリンク解除する必要があります。
> - Jira サイトを切断するには、まず、そのアカウントからリンクされたすべての Jira プロジェクトのリンクを解除する必要があります。
>
> 詳細については、「での GitHub リポジトリ、Bitbucket リポジトリ、Jira プロジェクトのリンク解除 CodeCatalyst」を参照してください。

GitHub プロジェクト、Bitbucket ワークスペース、または Jira サイトを切断するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. CodeCatalyst スペースに移動します。
3. 次のいずれかを実行して、スペースにインストールされている拡張機能のリストを表示します。

   a. 設定 を選択し、インストールされた拡張機能 を選択します。

b. トップメニュー 
タログアイコンを選択します。

4. 設定する拡張機能の 1 つとして、リポジトリ 、Bitbucket GitHub リポジトリ 、または Jira ソフトウェアを選択します。
5. 設定を選択したサードパーティーの拡張機能に応じて、次のいずれかを実行します。

   - GitHub リポジトリ： GitHub アカウントへの接続を解除します。

     「接続された GitHub アカウント」タブで、切断する GitHub アカウントを選択し、 GitHub 「アカウントを切断する」を選択します。
   - Bitbucket リポジトリ : Bitbucket ワークスペースに接続しません。

     Connected Bitbucket ワークスペースタブで、切断する Bitbucket ワークスペースを選択し、Bitbucket ワークスペースの切断 を選択します。
   - Jira ソフトウェア : Jira サイトへの接続を解除します。

     Connected Jira sites タブで、切断する Jira サイトを選択し、次に Disconnect Jira site を選択します。
6. 「切断」ダイアログボックスで、アカウントを切断した場合の影響を確認します。
7. テキスト入力フィールドに切断と入力し、切断を選択します。

での GitHub リポジトリ、Bitbucket リポジトリ、Jira プロジェクトのリンク CodeCatalyst

GitHub または Bitbucket リポジトリを使用するか、Jira プロジェクトを管理する前に、リポジトリまたはプロジェクトが属するサードパーティーのソースを CodeCatalyst スペースに接続する必要があります。詳細については、「 GitHub アカウント、Bitbucket ワークスペース、Jira サイトの接続 CodeCatalyst」を参照してください。

リンクされたリポジトリ GitHub または Bitbucket リポジトリは、ワークフローで使用できます。リンクされたリポジトリのイベントは、ワークフロー設定に応じて、コードの構築、テスト、デプロイを行うワークフローを開始します。リンクされたリポジトリ GitHub または Bitbucket リポジトリを使用するワークフローのワークフロー設定ファイルは、リンクされたリポジトリに保存されます。リンクされたリポジトリは、開発環境とともに使用して、リンクされたリポジトリ内のファイルを作

成、更新、削除することもできます。 GitHub または Bitbucket リポジトリを CodeCatalyst プロジェクトにリンクするには、GitHub リポジトリまたは Bitbucket リポジトリ拡張の詳細ページから、またはプロジェクト自体の Code のソースリポジトリビューから使用できます。

リンクされた Jira プロジェクトを使用して、問題を管理し、 CodeCatalyst プルリクエストを Jira 問題にリンクできます。プルリクエストの概要ステータスと、関連する CodeCatalyst ワークフローイベントのステータスが Jira の問題に反映されます。

Important

GitHub または Bitbucket リポジトリを寄稿者 としてリンクすることはできますが、サードパーティーリポジトリのリンクを解除できるのは、スペース管理者またはプロジェクト管理者 のみです。詳細については、「での GitHub リポジトリ、Bitbucket リポジトリ、Jira プロジェクトのリンク解除 CodeCatalyst」を参照してください。

Important

Jira プロジェクトを CodeCatalyst プロジェクトにリンクするには、スペース管理者または CodeCatalyst プロジェクト管理者である必要があります CodeCatalyst 。

Important

リポジトリ拡張をインストールすると、リンク先のリポジトリにコード CodeCatalyst のインデックスが作成され、 に保存されます CodeCatalyst。これにより、コードは で検索できるようになります CodeCatalyst。でリンクされたリポジトリを使用する場合のコードのデータ保護の詳細については CodeCatalyst、「Amazon CodeCatalyst ユーザーガイド」の「データ保護」を参照してください。

Note

- GitHub または Bitbucket リポジトリは、スペース内の 1 つの CodeCatalyst プロジェクトにのみリンクできます。
- 空のリポジトリ、アーカイブされたリポジトリ GitHub 、または Bitbucket リポジトリを CodeCatalyst プロジェクトで使用することはできません。

- プロジェクト内の CodeCatalyst リポジトリと同じ名前の GitHub または Bitbucket リポジトリをリンクすることはできません。
- GitHub リポジトリ拡張機能は GitHub Enterprise Server リポジトリと互換性がありません。
- Bitbucket リポジトリ拡張機能は Bitbucket データセンターリポジトリと互換性がありません。
- CodeCatalyst プロジェクトは 1 つの Jira プロジェクトにのみリンクできます。Jira プロジェクトは複数のプロジェクトにリンク CodeCatalyst できます。

トピック

接続されたサードパーティープロバイダーからのリソースのリンク

拡張の詳細ページから GitHub リポジトリ、Bitbucket リポジトリ、または Jira プロジェクトを CodeCatalyst プロジェクトにリンクするには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. CodeCatalyst スペースに移動します。
3. 次のいずれかを実行して、スペースにインストールされている拡張機能のリストを表示します。

   a. 設定 を選択し、インストールされた拡張機能 を選択します。

   b. トップメ

      ニュー
      タログアイコンを選択します。
4. リポジトリ 、Bitbucket GitHub リポジトリ 、または Jira ソフトウェア のいずれかの拡張機能に対して Configure を選択します。
5. 設定を選択したサードパーティーの拡張機能に応じて、次のいずれかを実行します。

   - GitHub リポジトリ: GitHub リポジトリをリンクします。

     1. リンクされた GitHub リポジトリ タブで、 GitHub リポジトリ のリンク を選択します。

2. GitHub アカウントドロップダウンから、リンクするリポジトリを含む GitHub アカウントを選択します。
3. GitHub リポジトリドロップダウンから、 CodeCatalyst プロジェクトにリンクするリポジトリを選択します。

> **Tip**
>
> リポジトリの名前がグレー表示になっている場合、そのリポジトリはスペース内の別のプロジェクトに既にリンクされているため、リンクできません。

4. （オプション) GitHub リポジトリのリストにリポジトリが表示されない場合は、 の Amazon CodeCatalyst アプリケーションでリポジトリアクセスが設定されていない可能性があります GitHub。接続されたアカウントの CodeCatalyst で使用できる GitHub リポジトリを設定できます。
   a. GitHub アカウントに移動し、設定 を選択し、アプリケーション を選択します。
   b. Installed GitHub Apps タブで、Amazon CodeCatalyst アプリケーションの設定を選択します。
   c. でリンクする GitHub リポジトリへのアクセスを設定するには、次のいずれかを実行します CodeCatalyst。
      - 現在および将来のすべてのリポジトリへのアクセスを提供するには、すべてのリポジトリ を選択します。
      - 特定のリポジトリへのアクセスを許可するには、リポジトリの選択のみ を選択し、リポジトリの選択 ドロップダウンを選択し、 でリンクを許可するリポジトリを選択します CodeCatalyst。
5. CodeCatalyst プロジェクトドロップダウンメニューから、 GitHub リポジトリをリンクする CodeCatalyst プロジェクトを選択します。
6. [Link (リンク)] を選択します。

で GitHub リポジトリを使用しなくなった場合は CodeCatalyst、プロジェクトからリポジトリの CodeCatalyst リンクを解除できます。リポジトリのリンクが解除されると、そのリポジトリのイベントはワークフロー実行を開始せず、 CodeCatalyst 開発環境でそのリポジトリを使用することはできません。詳細については、「での GitHub リポジトリ、Bitbucket リポジトリ、Jira プロジェクトのリンク解除 CodeCatalyst」を参照してください。

- Bitbucket リポジトリ : Bitbucket リポジトリをリンクします。

1. Linked Bitbucket リポジトリタブで、Link Bitbucket リポジトリ を選択します。
2. Bitbucket ワークスペースドロップダウンから、リンクするリポジトリを含む Bitbucket ワークスペースを選択します。
3. Bitbucket リポジトリドロップダウンから、 CodeCatalyst プロジェクトにリンクするリポジトリを選択します。

> **Tip**
>
> リポジトリの名前がグレー表示になっている場合、そのリポジトリはスペース内の別のプロジェクトに既にリンクされているため、リンクできません。

4. CodeCatalyst プロジェクトのドロップダウンメニューから、Bitbucket リポジトリをリンクする CodeCatalyst プロジェクトを選択します。
5. [Link (リンク)] を選択します。

で Bitbucket リポジトリを使用しなくなった場合は CodeCatalyst、プロジェクトから CodeCatalyst リンクを解除できます。リポジトリのリンクが解除されると、そのリポジトリのイベントはワークフロー実行を開始せず、 CodeCatalyst 開発環境でそのリポジトリを使用することはできません。詳細については、「での GitHub リポジトリ、Bitbucket リポジトリ、Jira プロジェクトのリンク解除 CodeCatalyst」を参照してください。

- Jira ソフトウェア: Jira プロジェクトをリンクします。

1. Linked Jira projects タブで、Link Jira project を選択します。
2. Jira サイトのドロップダウンメニューから、リンクするプロジェクトを含む Jira サイトを選択します。
3. Jira プロジェクトのドロップダウンメニューから、プロジェクトにリンクする CodeCatalyst プロジェクトを選択します。
4. CodeCatalyst プロジェクトのドロップダウンメニューから、Jira CodeCatalyst プロジェクトにリンクするプロジェクトを選択します。
5. [Link (リンク)] を選択します。

Jira プロジェクトが CodeCatalyst プロジェクトにリンクされると、 CodeCatalyst 問題へのアクセスは完全に無効になり、 CodeCatalyst ナビゲーションペインの問題は Jira プロジェクトにリンクする Jira 発行項目に置き換えられます。

で Jira プロジェクトを使用しない場合は CodeCatalyst、プロジェクトからリンクを解除できます CodeCatalyst。Jira プロジェクトがリンク解除されると、Jira の問題はプロジェクトで CodeCatalyst利用できなくなり、 CodeCatalyst 問題は再び問題プロバイダーになります。詳細については、「での GitHub リポジトリ、Bitbucket リポジトリ、Jira プロジェクトのリンク解除 CodeCatalyst」を参照してください。

GitHub または Bitbucket リポジトリを、 CodeCatalyst プロジェクトのソースリポジトリページからプロジェクトにリンクするには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. CodeCatalyst プロジェクトに移動します。
3. ナビゲーションペインで [コード] を選択してから、[ソースリポジトリ] を選択します。
4. リポジトリの追加 を選択し、リポジトリのリンク を選択します。
5. リポジトリプロバイダードロップダウンメニューから、サードパーティーのリポジトリプロバイダー GitHubまたは Bitbucket のいずれかを選択します。
6. リンクを選択したサードパーティーのリポジトリプロバイダーに応じて、次のいずれかを実行します。

   - GitHub リポジトリ: GitHub リポジトリをリンクします。

     1. GitHub アカウントのドロップダウンメニューから、リンクするリポジトリを含む GitHub アカウントを選択します。
     2. GitHub リポジトリのドロップダウンメニューから、 CodeCatalyst プロジェクトをリンクする GitHub アカウントを選択します。
     3. （オプション) GitHub リポジトリのリストにリポジトリが表示されない場合は、 の Amazon CodeCatalyst アプリケーションでリポジトリアクセスが設定されていない可能性があります GitHub。接続されたアカウントの CodeCatalyst で使用できる GitHub リポジトリを設定できます。

        a. GitHub アカウントに移動し、設定 を選択し、アプリケーション を選択します。

        b. Installed GitHub Apps タブで、Amazon CodeCatalyst アプリケーションの設定を選択します。

        c. でリンクする GitHub リポジトリへのアクセスを設定するには、次のいずれかを実行します CodeCatalyst。

- 現在および将来のすべてのリポジトリへのアクセスを提供するには、すべてのリポジトリ を選択します。
- 特定のリポジトリへのアクセスを許可するには、リポジトリの選択のみ を選択し、リポジトリの選択 ドロップダウンを選択し、 でリンクを許可するリポジトリを選択します CodeCatalyst。

- Bitbucket リポジトリ: Bitbucket リポジトリをリンクします。

  1. Bitbucket ワークスペースのドロップダウンメニューから、リンクするリポジトリを含む Bitbucket ワークスペースを選択します。
  2. Bitbucket リポジトリのドロップダウンメニューから、 CodeCatalyst プロジェクトをリンクする Bitbucket リポジトリを選択します。

> **Tip**
>
> リポジトリの名前がグレー表示になっている場合、そのリポジトリは Amazon の別のプロジェクトに既にリンクされているため、リンクできません CodeCatalyst。

7. [Link (リンク)] を選択します。

で GitHub または Bitbucket リポジトリを使用しなくなった場合は CodeCatalyst、プロジェクトから CodeCatalyst リンクを解除できます。リポジトリのリンクが解除されると、そのリポジトリのイベントはワークフロー実行を開始せず、 CodeCatalyst 開発環境でそのリポジトリを使用することはできません。詳細については、「での GitHub リポジトリ、Bitbucket リポジトリ、Jira プロジェクトのリンク解除 CodeCatalyst」を参照してください。

GitHub または Bitbucket リポジトリを CodeCatalyst プロジェクトにリンクしたら、 CodeCatalyst ワークフローや開発環境で使用できます。リンクされたリポジトリは、Amazon Q デベロッパー、ブループリントなどでも使用できます。詳細については、「サードパーティーのリポジトリイベント後にワークフローを自動的に開始する」および「開発環境の作成」を参照してください。

Jira プロジェクトを CodeCatalyst プロジェクトにリンクし、プルリクエストをリンクすると、 の更新 CodeCatalyst が Jira プロジェクトに反映されます。プルリクエストを Jira の問題にリンクする方法の詳細については、「」を参照してくださいJira の問題の CodeCatalyst プルリクエストへのリンク。Jira での CodeCatalyst イベントの表示の詳細については、「」を参照してくださいJira の問題での CodeCatalyst イベントの表示。

CodeCatalyst プロジェクト作成時のサードパーティーリポジトリの へのリンク

新しい CodeCatalyst プロジェクトを作成するときに、 GitHub または Bitbucket リポジトリを新しい CodeCatalyst プロジェクトにリンクできます。詳細については、「リンクされたサードパーティーリポジトリを使用したプロジェクトの作成」を参照してください。

での GitHub リポジトリ、Bitbucket リポジトリ、Jira プロジェクトのリンク解除 CodeCatalyst

GitHub リポジトリまたは Bitbucket リポジトリを使用したり、 で Jira プロジェクトを管理したりする必要がなくなった場合は CodeCatalyst、 CodeCatalyst プロジェクトからリポジトリまたはプロジェクトのリンクを解除できます。

GitHub または Bitbucket リポジトリのリンクを解除しても、リポジトリは削除されず、変更も行われません。リンクされたリポジトリに保存されているワークフロー設定ファイルは削除されません。ただし、 GitHub または Bitbucket リポジトリのリンクを解除すると、そのリポジトリのイベントはワークフロー実行を開始せず、開発環境でリポジトリを使用することはできません。または Bitbucket リポジトリを GitHub CodeCatalyst プロジェクトにリンク解除するには、GitHub リポジトリまたは Bitbucket リポジトリ拡張の詳細ページから、またはプロジェクト自体の Code のソースリポジトリビューから行います。

Jira プロジェクトのリンクを解除しても、計画項目や開発情報などのプロジェクトは削除されず、変更も行われません。ただし、Jira プロジェクトのリンクを解除すると、プロジェクトの Jira の問題は CodeCatalyst プロジェクトにリンクできなくなり、 CodeCatalyst 問題は再び問題プロバイダーになります。

> **Important**
>
> GitHub リポジトリまたは Bitbucket リポジトリを CodeCatalyst プロジェクトからリンク解除するには、スペース管理者またはプロジェクト管理者である必要があります。

プロジェクト内の GitHub リポジトリ、Bitbucket リポジトリ、または Jira プロジェクトを拡張機能の詳細ページ CodeCatalyst からリンク解除するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. CodeCatalyst スペースに移動します。

3. 次のいずれかを実行して、スペースにインストールされている拡張機能のリストを表示します。

   a. 設定 を選択し、インストールされた拡張機能 を選択します。

   b. トップメ

      ニュー
      タログアイコンを選択します。

4. 設定する拡張の 1 つとして、GitHub リポジトリ 、Bitbucket リポジトリ 、または Jira ソフトウェア を選択します。

5. 設定を選択したサードパーティーの拡張機能に応じて、次のいずれかを実行します。

   - GitHub リポジトリ: GitHub リポジトリのリンクを解除します。

     GitHub リポジトリタブで、リンクを解除する GitHub リポジトリを選択し、リポジトリのリンク解除 GitHub を選択します。

   - Bitbucket リポジトリ: Bitbucket リポジトリのリンクを解除します。

     Bitbucket リポジトリタブで、リンクを解除する Bitbucket リポジトリを選択し、Bitbucket リポジトリ のリンク解除を選択します。

   - Jira ソフトウェア: Jira プロジェクトのリンクを解除します。

     Jira プロジェクトタブで、リンクを解除する Jira プロジェクトを選択し、Jira プロジェクトのリンク解除を選択します。

6. Unlink ダイアログボックスで、リポジトリのリンクを解除した場合の影響を確認します。

7. テキスト入力フィールドに unlink と入力し、Unlink を選択します。

CodeCatalyst プロジェクト内の GitHub または Bitbucket リポジトリをソースリポジトリページからリンク解除するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。

2. CodeCatalyst プロジェクトに移動します。

3. ナビゲーションペインで [コード] を選択してから、[ソースリポジトリ] を選択します。

4. リンクを解除するリポジトリのラジオボタンを選択し、リポジトリ のリンク解除を選択します。

5. ダイアログボックスの情報を確認します。手順に従って、リンク解除を選択してリポジトリのリンクを解除します。

でのサードパーティーリポジトリの表示と Jira の問題の検索 CodeCatalyst

GitHub または Bitbucket リポジトリをリンクした後、 でそれらを表示 CodeCatalyst して、リソースを確認および設定できます。リンクされた Jira の問題は、 で検索することもできます CodeCatalyst。

トピック

でのサードパーティーリポジトリの表示 CodeCatalyst

リンクされたリポジトリ GitHub または Bitbucket リポジトリは、プロジェクトのソースリポジトリのリスト、またはリポジトリまたは Bitbucket GitHub リポジトリ拡張の詳細ページから表示できます。リポジトリのリストからそれらを選択しても、 では開きません CodeCatalyst。代わりに、サードパーティーのリポジトリプロバイダーで を開き、リンクされたリポジトリでコードを表示して操作できます。

でリンクされたリポジトリ GitHub または Bitbucket リポジトリを表示するには CodeCatalyst

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. CodeCatalyst プロジェクトに移動します。
3. ナビゲーションペインで [コード] を選択してから、[ソースリポジトリ] を選択します。

拡張機能の詳細ページからリンクされたリポジトリ GitHub または Bitbucket リポジトリを表示するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. CodeCatalyst スペースに移動し、インストールされた拡張機能タブを選択します。
3. 表示するサードパーティーのリポジトリに応じて、次のいずれかを実行します。

    - リポジトリ GitHub で、「 の設定」を選択し、「リンクされた GitHub リポジトリ」を選択して、 CodeCatalyst スペース内の CodeCatalyst プロジェクトに接続されているすべての GitHub リポジトリを表示します。

- Bitbucket リポジトリ で、 の設定 を選択し、リンクされた Bitbucket リポジトリ を選択して、 CodeCatalyst スペース内の CodeCatalyst プロジェクトに接続されているすべての Bitbucket リポジトリを表示します。

CodeCatalyst プロジェクトにリンクされている GitHub または Bitbucket リポジトリがリストに表示されます。 GitHub または Bitbucket リポジトリを選択して、サードパーティーのリポジトリプロバイダーのファイルを表示および編集します。

> **Note**
>
> ワークフローがソースアクションで GitHub または Bitbucket リポジトリを使用する場合、ビジュアルエディタまたは の YAML エディタでワークフロー YAML に加えた変更は、自動的にコミットされ、サードパーティーのリポジトリにプッシュ CodeCatalyst されます。

での Jira の問題の検索 CodeCatalyst

Jira プロジェクトをリンクした後、 CodeCatalyst グローバル検索バーを使用して、リンクされた Jira プロジェクトで問題を検索できます。プルリクエストから問題にリンク CodeCatalyst している間に、 で Jira の問題を検索することもできます。Jira の問題の CodeCatalyst プルリクエストへのリンクの詳細については、「」を参照してくださいJira の問題の CodeCatalyst プルリクエストへのリンク。

リンクされた Jira プロジェクトで Jira の問題を検索するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. CodeCatalyst プロジェクトに移動します。
3. グローバル検索バーで、リンクされた Jira プロジェクトで、プルリクエストにリンクする問題または Jira の問題を検索します。

サードパーティーのリポジトリイベント後にワークフローを自動的に開始する

リンクされたリポジトリ GitHub または Bitbucket リポジトリをワークフローのソースとして使用できます。リンクされたリポジトリ GitHub または Bitbucket リポジトリ内の指定されたブランチへの変更によって、ワークフローの実行が自動的に開始されます。

ワークフローは、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) システムの一部としてコードを構築、テスト、デプロイする方法を説明する自動化された手順です。ワークフローは、ワークフローの実行中に実行する一連のステップまたはアクションを定義します。ワークフローは、ワークフローを開始するイベント、またはトリガー も定義します。ワークフローを設定するには、CodeCatalyst コンソールのビジュアルまたは YAML エディタ を使用してワークフロー定義ファイルを作成します。

> **Tip**
>
> プロジェクトでワークフローを使用する方法を簡単に確認するには、設計図 を使用してプロジェクトを作成します。各ブループリントは、レビュー、実行、および実験できる機能するワークフローをデプロイします。

リンクされたリポジトリ GitHub または Bitbucket リポジトリを使用するようにワークフローを設定すると、ワークフロー設定ファイルはそのリポジトリ GitHub または Bitbucket リポジトリに保存されます。ワークフロー設定は、ワークフロー名、トリガー、リソース、アーティファクト、アクションを定義する YAML ファイルです。ワークフロー設定ファイルの詳細については、「」を参照してくださいワークフロー YAML 定義。

ワークフロー設定ファイルは、 GitHub または Bitbucket リポジトリの `./codecatalyst/workflows/` ディレクトリにある必要があります。

ワークフローエディタを使用して、ワークフローを作成および設定できます。詳細については、「ワークフローの開始方法」および「ワークフローをソースリポジトリに接続する」を参照してください。

ワークフロー実行を開始するためのトリガーの追加

コードが GitHub または Bitbucket リポジトリの指定されたブランチにプッシュされたときに実行を自動的に開始するように CodeCatalyst ワークフローを設定できます。ワークフローの実行を自動的に開始するには、ワークフロー設定ファイルの `Triggers`セクションにトリガーを追加します。

例: シンプルなコードプッシュトリガー

次の例は、ソースリポジトリ内のブランチにコードがプッシュされるたびにワークフロー実行を開始するトリガーを示しています。

```
Triggers:
```

```
  - Type: PUSH
```

例: シンプルなプルリクエストトリガー

次の例は、ソースリポジトリ内のブランチに対してプルリクエストが作成されるたびにワークフロー実行を開始するトリガーを示しています。

```
Triggers:
  - Type: PULLREQUEST
    Events:
      - OPEN
```

詳細については、「トリガーを使用したワークフローの自動実行の開始」を参照してください。

GitHub Enterprise Cloud を使用した での IP アクセスの制限

ルール GitHub または設定を設定することで、IP アドレスに基づいて または Bitbucket リポジトリへのアクセスを制限できます。これを行うには、サードパーティープロバイダーの設定またはアクセスコントロール機能を使用します。

使用しているサードパーティーのリポジトリプロバイダーに応じて、次のいずれかを参照してください。

- Amazon CodeCatalyst GitHub リポジトリ拡張機能は、 GitHub Enterprise Cloud の IP アクセス制限 と互換性があります。特定の IP アドレスへのアクセスを制限するように GitHub Enterprise Cloud 組織を設定する場合、 GitHub アプリケーションが許可リスト を設定できるようにすることもできます。これにより、 が IP アドレスを に自動的に CodeCatalyst 登録できるようになります GitHub。または、 CodeCatalyst IP アドレス を手動で追加することもできます。
- Amazon CodeCatalyst Bitbucket リポジトリ拡張機能は、Bitbucket Cloud Premium アクセス制限 と互換性があります。特定の IP アドレスへのアクセスを制限するように Bitbucket Cloud Premium ワークスペースを設定する場合、一連の IP アドレスの IP アドレスまたはネットワークブロックを許可リスト に追加することもできます。

CodeCatalyst IP アドレスがサードパーティーリポジトリの許可リストにない場合、Amazon CodeCatalyst アプリはサードパーティーリポジトリにアクセスできません。詳細については、「サードパーティーのリポジトリ拡張で使用される IP アドレス」を参照してください。

サードパーティーのリポジトリ拡張で使用される IP アドレス

以下の IP アドレスは、サードパーティーの拡張機能がサードパーティーのリソースにアクセスするために使用します。

- GitHub リポジトリ：

```
us-west-2
  52.32.242.246
  54.148.176.49
  35.164.118.94
eu-west-1
  34.241.64.10
  34.246.255.80
  3.248.38.7
```

- Bitbucket リポジトリ：

```
us-west-2
  35.160.210.199
  54.71.206.108
  54.71.36.205
eu-west-1
  34.242.64.82
  52.18.37.201
  54.77.75.62
```

ワークフローが失敗した場合にサードパーティーのプルリクエストマージをブロックする

GitHub リポジトリを にリンクした後 CodeCatalyst、プルリクエストの CodeCatalyst ワークフローを追加できます。1 つ以上のワークフロー実行が特定のコミットで実行でき、 の各ワークフローの実行ステータス CodeCatalyst は、 GitHub または Bitbucket のコミットステータスの一部としても反映されます。新しいコミットがプッシュされると、その新しいコミット GitHub の新しいワークフロー実行ステータスが に反映されます。コミットに対してワークフローを再度実行すると、新しいワークフロー実行ステータスによって、そのコミットとワークフローの以前のステータスが上書きされます。

でブランチ保護ルールを設定 GitHub して、最新のコミットのワークフロー実行ステータスが失敗した場合にプルリクエストのマージをブロックできます。ブランチ保護ルールでは、最新のコミットのステータスは、 でプルリクエストをマージする機能に影響します GitHub。詳細については、GitHubのドキュメント「ステータスチェックについて」および「保護されたブランチについて」を参照してください。ワークフローの詳細については、ワークフローの実行「」および「」を参照してくださいトリガーを使用したワークフローの自動実行の開始。

Jira の問題の CodeCatalyst プルリクエストへのリンク

CodeCatalyst ソースリポジトリで作成されたプルリクエストを Jira の問題にリンクできます。Jira 問題をリンクすると、その問題はプルリクエストのプロパティとして表示されます。その結果、プルリクエストイベント、ワークフローイベント、デプロイイベントが Jira に送信され、Jira の問題に追加されます。プルリクエストは、1 つ以上の Jira 問題にリンクできます。リンクできるのは、CodeCatalyst ソースリポジトリにあるプルリクエストのみです。 などのサードパーティーリポジトリにあるプルリクエストはリンクできません GitHub。Jira 問題をプルリクエストにリンクする前に、Jira プロジェクトを CodeCatalyst プロジェクトにリンクする必要があります。Jira プロジェクトを CodeCatalyst プロジェクトにリンクする方法の詳細については、「」を参照してくださいでの GitHub リポジトリ、Bitbucket リポジトリ、Jira プロジェクトのリンク CodeCatalyst。

> **Note**
>
> CodeCatalyst プロジェクトに 2 つのブランチがあるソースリポジトリがないと、プルリクエストを作成できません。プルリクエストの詳細については、「 でのプルリクエストの使用 CodeCatalyst」を参照してください。

Jira 問題を CodeCatalyst プルリクエストにリンクするには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. CodeCatalyst プロジェクトに移動します。
3. ナビゲーションペインで、コード を選択し、プルリクエスト を選択します。
4. プルリクエストの作成 を選択して、プルリクエストの詳細を入力します。
5. ソースリポジトリのドロップダウンメニューから、プルリクエストをリンクするソースリポジトリを選択します。
6. ソースブランチドロップダウンメニューから、レビューする変更を含むブランチを選択します。

7. 送信先ブランチのドロップダウンメニューから、レビューされた変更をマージするブランチを選択します。
8. プルリクエストタイトルのテキスト入力フィールドに、プルリクエストのタイトルを入力します。
9. Jira の問題のリンク - オプションフィールドを選択し、ドロップダウンを選択して、リンクされた Jira プロジェクトから追加する Jira の問題を検索します。
10. プルリクエストに追加する Jira の問題を選択します。
11. 作成 を選択してプルリクエストを作成します。

Jira の問題を CodeCatalyst プルリクエストにリンクすると、プルリクエストの概要が表示されます。概要には、ワークフローの実行、リンクされた問題、必要なレビューワー、オプションのレビューワー、作成者が含まれます。

> **Note**
>
> Jira の問題に関連する情報によって作成された担当者と作成者は、 では使用できません CodeCatalyst。

プルリクエストをリンクすると、同期された CodeCatalyst プロジェクトと Jira プロジェクトにより、 からの更新 CodeCatalyst が Jira プロジェクトに反映されます。リンクされたプルリクエストのステータスと、プルリクエストに関連するワークフローイベントは、Jira で表示すると Jira の問題に表示されます。Jira での CodeCatalyst イベントの表示の詳細については、「」を参照してくださいJira の問題での CodeCatalyst イベントの表示。

Jira の問題での CodeCatalyst イベントの表示

CodeCatalyst プロジェクトと Jira プロジェクトがリンクされている場合、プルリクエストの概要ステータスおよび関連する CodeCatalyst ワークフローイベントのステータスが Jira の問題に反映されます。例えば、 でプルリクエストを閉じたりマージしたりすると CodeCatalyst、ステータスの更新は Jira の問題に反映されます。 CodeCatalyst CodeCatalyst プルリクエストに関連するワークフロー CI/CD イベントは同期されるため、ワークフローの実行が成功すると Jira の問題にも送信されます。

Jira の問題の CodeCatalyst イベントを表示するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。

2. CodeCatalyst プロジェクトに移動します。
3. CodeCatalyst ナビゲーションペインで、コード を選択し、プルリクエスト を選択し、Jira プロジェクトで表示する Jira 問題を含むプルリクエストを選択します。
4. 追加情報ペインで、Jira プロジェクトで表示する Jira 問題を選択します。
5. Jira プロジェクトの詳細ペインで、開発 にリストされているプルリクエストを選択して、プルリクエストの詳細を表示します。
6. (オプション) 最新のビルドを表示するには、ビルドタブを選択します。
7. (オプション) 開発ステータスを表示するには、デプロイタブを選択します。

でコード、問題、プロジェクト、ユーザーを検索する CodeCatalyst

で検索バーまたは専用の検索結果ウィンドウを使用して CodeCatalyst 、コード、問題、プロジェクト、およびユーザーを検索します CodeCatalyst。

名前、説明、ステータスなどのクエリを検索バーに入力することで、スペースやプロジェクト全体のリソースを検索できます。検索クエリ言語を使用して検索クエリを絞り込むこともできます。

トピック

- 検索クエリの改良
- 検索を使用する際の考慮事項
- 検索可能なフィールドのリファレンス

検索するには

1. 上部のナビゲーションバーの検索バーに、検索クエリを入力します。
2. (オプション) の検索クエリ言語を使用して CodeCatalyst検索クエリを絞り込みます。詳細については、「検索クエリの改良」を参照してください。
3. 次のいずれかを行います。
   - 現在使用しているプロジェクト内のリソースを検索するには、このプロジェクト を選択します。
   - 現在あるスペース内のすべてのプロジェクト内のリソースを検索するには、このスペースを選択します。
4. 次のいずれかを実行して、専用の検索結果ウィンドウで検索結果を表示します。
   - クイック検索結果ウィンドウの下部で、すべての検索結果を project-name で表示 | space-name を選択して、すべての検索結果を表示します。
   - Enter キーを押して、すべての検索結果を表示します。

> **Tip**
>
> @ 記号に続けて表示名またはユーザー名を使用して、プルリクエストのコメントまたは説明、または問題のコメントまたは説明に他のプロジェクトユーザーに言及します。@ 記号の

後に問題やコードファイルの名前を付けることで、問題やコードファイルなどのリソースにリンクすることもできます。

検索クエリの改良

検索後に探しているものが見つからない場合は、 CodeCatalystの特殊なクエリ言語を使用して検索を絞り込むことができます。個々のフィールドには文字数制限はありませんが、クエリ全体の文字数は 1,024 文字に制限されています。

トピック

タイプによる絞り込み

検索の範囲を特定のタイプの情報に絞り込むには、検索*type:result-type*に を含めます。ここで*######*は code、、project、または issueですuser。

例:

- `type:code AND java` – 「java」を含むコード関連のフィールドにコード結果を表示します。

  詳細については、「コードフィールド」を参照してください。
- `type:issue AND Bug` – 「バグ」を含む問題関連のフィールドに問題の結果を表示します。

  詳細については、「問題フィールド」を参照してください。
- `type:user AND MaryMajor` – MaryMajor「」を含むユーザー関連フィールドにユーザー結果を表示します。

  詳細については、「ユーザーフィールド」を参照してください。
- `type:project AND Datafeeder` – 「Datafeeder」を含むプロジェクト結果を表示します。

  詳細については、「プロジェクトフィールド」を参照してください。

フィールドによる絞り込み

検索の範囲を特定のフィールドに絞り込むには、検索*field-name:query*に を含めます。ここで、*######*は title、description、、 username projectなどで、*###*は検索するテキストです。フィールドのリストについては、「」を参照してください検索可能なフィールドのリファレンス。括弧を使用して複数のクエリを検索できます。

例:

- `title:bug` – タイトルに「バグ」が含まれている結果を表示します。
- `username:John` – ユーザー名に「John」が含まれている結果を表示します。
- `project:DataFeeder` - プロジェクトDataFeeder「」に結果を表示します。クエリでは大文字と小文字は区別されません。
- `description:overview` – 説明に「概要」が含まれている結果を表示します。

ブール演算子による絞り込み

検索フレーズの制約を指定するには、ブール演算子 AND、OR、および を使用できますNOT。複数のフレーズを一覧表示すると、 CodeCatalyst OR はデフォルトでそれらを に結合します。検索フレーズは括弧を使用してグループ化できます。

- `exception AND type:code` – 「例外」のコード結果のみを表示します。
- `path:README.md AND repo:ServerlessAPI` – リポジトリの名前が「ServerlessAPI」である README.md」のパスの結果を表示します。 ServerlessAPI
- `buildspec.yml AND (repo:ServerlessAPI OR ServerlessWebApp)` – リポジトリが ServerlessAPI」または「」である「buildspec.ymlServerlessWebApp」の結果を表示します。
- `path:java NOT (path:py OR path:ts)` – パスに「java」が含まれているが、「py」や「ts」が含まれていない結果を表示します。

プロジェクトによる改良

検索の範囲を特定のプロジェクトに絞り込むには、検索*project:name AND query*に を含めます。ここで、*name* は検索対象のプロジェクト、*Query* は検索対象のコンテンツです。

- `project:name AND query` – パスにクエリとプロジェクト名が含まれている結果を表示します。

検索を使用する際の考慮事項

コンテンツの更新の遅延 – 名前の変更や問題の再割り当てなどのコンテンツの更新が検索結果に反映されるまでに数分かかることがあります。コードベースの移行などの大規模な更新は、検索結果に表示されるまでに時間がかかる場合があります。

特殊文字のエスケープ – 検索クエリでは、次の特殊文字に特別な考慮事項が必要です: `+ - & & || ! ( ) { } [ ] ^ " ~ * ? : \`。特殊文字はクエリには影響しません。特殊文字を削除またはエスケープする必要があります。キャラクターをエスケープするには、キャラクターの前にバックスラッシュ (\) を追加します。例えば、検索クエリ [Feature] は Feature または \[Feature\] のいずれかである必要があります。

検索の絞り込み — 検索では大文字と小文字は区別されません。すべての小文字で検索すると、クエリがケース変更時に単語を分割できなくなります。例えば、 `MyService`と のみをクエリするには`MyService`、 `my`または のみを含む結果を避けるため`myservice`にクエリすることを検討してください`service`。

検索は、デフォルトで OR 単位の結合を使用して単語と単語の一部を結合します。例えば、 `new function`は `new`と の両方を含む結果を返す`function`ことができ、 `new`または のみを含む結果も返すことができます`function`。後者を避けるには、複数の単語を と組み合わせます`AND`。例えば、を検索できます`new AND function`。

デフォルトブランチ – 検索では、ソースリポジトリのデフォルトブランチに対する最新のコミットからのコード結果のみが返されます。他のブランチやコミットでコードを検索するには、リポジトリのクローンをローカルで作成するか、開発環境 でブランチを開くか、 CodeCatalyst UI でブランチと詳細を表示することを検討してください。デフォルトのブランチを変更すると、検索によって検出可能なファイルが更新される。詳細については、「リポジトリのデフォルトブランチの管理」を参照してください。

検索可能なフィールドのリファレンス

CodeCatalyst は、検索クエリを入力するときに次のフィールドを検索します。エイリアスは、高度なクエリ言語でフィールドを参照するために使用できる別の名前です。

コードフィールド

フィールド	エイリアス	説明
branchName	ブランチ	コードファイルが置かれているブランチの名前。
コード	該当なし	検索に一致したソースコードの一部を示すコードスニペット形式のコードコンテンツに関する情報。
commitId	該当なし	返されたコードファイルが最後に更新されたコミットのコミット ID。で指定されたブランチ名の末尾にあるコミット ID は、 でも でも でもないこともありますbranchName 。
commitMessage	該当なし	コードファイルが最後に更新されたコミットのコミットメッセージ。で指定されたブランチ名の末尾にあるコミットメッセージであるまたは は、コミットメッセージではない場合がありますbranchName 。コミットメッセージが指定されていない場合、この値は空の文字列になります。
filePath	パス	このコードファイルのファイルパス。
lastUpdatedBy	該当なし	CodeCatalyst コードファイルを最後に更新した ユーザー。ユーザー名が使用できない場合、この値は Git 設定ファイ

フィールド	エイリアス	説明
		ルで設定されたユーザーの E メールアドレスになります。
lastUpdatedByID	該当なし	コードファイルを最後に更新したユーザーのシステム生成の一意の ID。ユーザー ID が使用できない場合、この値はユーザーの E メールアドレスである可能性があります。
lastUpdatedTime	該当なし	検索データがコードファイル (協定世界時 (UTC) タイムスタンプ) を含むコミットで最後に更新された時刻。
projectId	該当なし	システムによって生成されたプロジェクトの一意の ID。
projectName	projectNames、プロジェクト	コードファイルがコミットされたソースリポジトリを含むプロジェクトの名前を表示します。
repositoryId	repoId	システムによって生成されたソースリポジトリの一意の ID。
repositoryName (リポジトリネーム)	リポジトリ、リポジトリ	コードファイルがコミットされたソースリポジトリの名前を表示します。

問題フィールド

フィールド	エイリアス	説明
assigneeIds	assigneeId	問題に割り当てられたユーザーのシステム生成IDs。
担当者	担当者	問題に割り当てられたユーザーのユーザー名。
createdBy	該当なし	問題を作成したユーザーの名前を表示します。
createdById	該当なし	問題を作成したユーザーのシステム生成の一意の ID。
createdTime	該当なし	問題が作成された時刻 (協定世界時 (UTC) タイムスタンプ) 。
説明	該当なし	問題の説明。
isArchived	archived	アーカイブされた状態で問題を作成するかどうかを示すブール値。
isBlocked	ブロック済み	問題がブロック済みとしてマークされているかどうかを示すブール値。
labelIds	labelId	問題に対するラベルのシステム生成の一意の IDs。
lastUpdatedBy	該当なし	問題の最終更新者である の使用名を表示します。
lastUpdatedByID	該当なし	最後に問題を更新したユーザーのシステム生成の一意の ID。

フィールド	エイリアス	説明
lastUpdatedTime	該当なし	問題が最後に更新された時刻 (協定世界時 (UTC) タイムスタンプ) 。
priority	該当なし	問題が割り当てられているかどうかの優先度。
projectId	該当なし	システムによって生成されたプロジェクトの一意の ID。
projectName	projectNames、プロジェクト	この問題が見つかったプロジェクト。
shortId	該当なし	問題の短縮された自動増分識別子。
status	該当なし	問題がボードのバックログまたは列にあるかどうかを示す問題のステータス。
statusId	該当なし	ステータスのシステム識別子。
title	該当なし	問題のタイトル。

プロジェクトフィールド

フィールド	エイリアス	説明
説明	該当なし	プロジェクトの説明。
lastUpdatedTime	該当なし	プロジェクトメタデータが最後に更新された時刻 (協定世界時 (UTC) タイムスタンプ) 。

フィールド	エイリアス	説明
projectName	プロジェクト	スペース内のプロジェクトの名前。
projectPath	該当なし	プロジェクトの作成時に定義される、プロジェクトの URL ルーティング可能な名前。プロジェクト名を必要とする URLs で使用されます。

ユーザーフィールド

フィールド	エイリアス	説明
displayName	該当なし	でユーザーに使用される名前 CodeCatalyst。表示名は一意ではありません。
email	該当なし	ユーザーの E メールアドレス。
lastUpdatedTime	該当なし	ユーザーメタデータが最後に更新された時刻 (協定世界時 (UTC) タイムスタンプ) 。
userName	username	にサインアップしたときにユーザーが選択したユーザー名 CodeCatalyst。表示名とは異なり、ユーザー名は変更できません。

Amazon のトラブルシューティング CodeCatalyst

以下の情報は、の一般的な問題のトラブルシューティングに役立ちます。 CodeCatalystAmazon CodeCatalyst ヘルスレポートを使用して、エクスペリエンスに影響を与える可能性のあるサービスの問題があるかどうかを判断することもできます。

トピック

アクセスに関する一般的な問題のトラブルシューティング

パスワードを忘れてしまいました

問題:AWSビルダー ID と Amazon に使用しているパスワードを忘れてしまいました CodeCatalyst。

解決方法:この問題を解決する最も簡単な方法は、パスワードをリセットすることです。

1. Amazon を開いて CodeCatalyst、メールアドレスを入力します。次に、[Continue] (続行する) を選択します。
2. [パスワードをお忘れですか?] を選択します。
3. パスワードを変更するためのリンクが記載されたメールが送信されます。受信トレイにメールが表示されない場合は、迷惑メールフォルダを確認してください。

Amazon CodeCatalyst の一部または全部がご利用いただけません

問題:コンソールに移動したり、 CodeCatalyst コンソールへのリンクをたどったりしましたが、エラーが表示されます。

解決方法:この問題の最も一般的な原因は、招待されていないプロジェクトやスペースへのリンクをたどったか、サービスに一般的な空き状況の問題があることです。Health レポートをチェックして、サービスに既知の問題がないかどうかを確認してください。そうでない場合は、プロジェクトまたはスペースにあなたを招待した人に連絡して、別の招待を依頼してください。まだプロジェクトやスペースに招待されていない場合は、サインアップして自分のスペースやプロジェクトを作成できます。

でプロジェクトを作成できない CodeCatalyst

問題:プロジェクトを作成したいのに、[プロジェクトを作成] ボタンが使用できないと表示されるか、エラーメッセージが表示されます。

解決方法:この問題の最も一般的な原因は、AWSスペース管理者ロールを持たないビルダー ID でコンソールにサインインしていることです。スペースにプロジェクトを作成するには、このロールが必要です。

このロールを持っているのにボタンが使用可能と表示されない場合は、サービスに一時的な問題がある可能性があります。ブラウザを更新して、もう一度試してください。

サポート問題のトラブルシューティング

Amazon AWS Support にアクセスするとエラーが出る CodeCatalyst

問題:AWS Supportfor Amazon CodeCatalyst オプションを選択すると、次のエラーメッセージが表示されます。

`Unable to assume role`

```
To access support cases, you must add the role
AWSRoleForCodeCatalystSupport to the AWS ##### that is the billing account
for the space.
```

考えられる解決方法:AWS アカウントスペースの請求先アカウントに必要なロールを追加します。スペースの請求アカウントとして指定されたアカウント

は、AWSRoleForCodeCatalystSupportAmazonCodeCatalystSupportAccessロールと管理ポリシーを使用します。詳細については、「アカウントとスペースのAWSRoleForCodeCatalystSupportロールの作成」を参照してください。

> **Note**
>
> AWSビルダー ID がサポートされるのは、認証に使用したエイリアスとアクセス権限に基づくリソースのみです。 CodeCatalystアカウントと請求のサポートは、スペース内のすべてのユーザーが利用できます。ただし、ビルダーがサポートを受けることができるのは、自分が権限を持っているリソースと情報だけです。 CodeCatalyst

自分のスペースのテクニカルサポートケースを作成できない

問題:自分のスペースのテクニカルサポートケースを作成できません。

解決策:スペース内のユーザーがテクニカルSupport ケースを作成できるようにするには、ビジネスSupport またはエンタープライズサポートプランをスペース請求アカウントに追加する必要があります。AWS Supportスペース管理者にスペース請求アカウントにプランを追加するよう依頼するか、https://repost.aws/ AWS にアクセスしてコミュニティに問い合わせてください。

サポートケースのアカウントが、自分のスペースに接続されなくなりました。 CodeCatalyst

問題:サポートケースのアカウントがスペースインに接続されなくなりました CodeCatalyst。

解決策:スペース管理者権限を持つユーザーがスペース請求アカウントを切り替えると、AWS Supportプランと関連するすべてのケースがスペースから切り離されます。AWS Support古いスペース請求アカウントに関連付けられているケースは、AWS Support for Amazon では表示されなくなります CodeCatalyst。その請求アカウントのルートユーザーは、から古いケースを表示して解決できます。また、他のユーザーが古いケースを閲覧して解決できるように IAM 権限を設定できます。AWS Management Console AWS Support古い Space 請求アカウントのテクニカルサポートを引き続き受けることはできませんがAWS Management Console、AWS Supportプランがキャンセルされるまでは他のサービスのテクニカルサポートを受けることができます。 CodeCatalyst

詳細については、『AWS Supportユーザーガイド』の「ケースの更新、解決、再開」を参照してください。

Amazon AWS のサービスAWS Support で別のサポートケースを開くことができません CodeCatalyst

問題:AWS のサービス別のフォームのサポートケースをオープンできません CodeCatalyst。AWS Support

考えられる解決策:AWS Supportfor CodeCatalyst からのみサポートケースを開くことができます CodeCatalyst。別のサービスAWS、Amazon、 CodeCatalyst またはその他のサードパーティのサービスにデプロイされたサービスまたはリソースのサポートが必要な場合は、AWS Management Consoleまたはサードパーティのサービスサポートチャネルを通じてケースを作成する必要があります。詳細については、『AWS Supportユーザーガイド』の「サポートケースの作成とケース管理」を参照してください。

Amazon CodeCatalyst の一部または全部がご利用いただけません

問題:コンソールに移動したり、 CodeCatalyst コンソールへのリンクをたどったりしましたが、エラーが表示されます。

解決方法:この問題の最も一般的な原因は、招待されていないプロジェクトやスペースへのリンクをたどったか、サービスに一般的な空き状況の問題があることです。Health レポートをチェックして、サービスに既知の問題がないかどうかを確認してください。そうでない場合は、プロジェクトまたはスペースにあなたを招待した人に連絡して、別の招待を依頼してください。まだプロジェクトやスペースに招待されていない場合は、サインアップして自分のスペースやプロジェクトを作成できます。

でプロジェクトを作成できない CodeCatalyst

問題:プロジェクトを作成したいのに、[プロジェクトを作成] ボタンが使用できないと表示されるか、エラーメッセージが表示されます。

解決方法:この問題の最も一般的な原因は、AWSスペース管理者ロールを持たないビルダー ID でコンソールにサインインしていることです。スペースにプロジェクトを作成するには、このロールが必要です。

このロールを持っているのにボタンが使用可能と表示されない場合は、サービスに一時的な問題がある可能性があります。ブラウザを更新して、もう一度試してください。

フィードバックを送信したい CodeCatalyst

問題:にバグが見つかったので CodeCatalyst 、フィードバックを送信したい。

考えられる解決方法:フィードバックは直接に送信できます CodeCatalyst。

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. ナビゲーションペインで [フィードバックを送信] を選択します。
3. ドロップダウンメニューからフィードバックの種類を選択し、フィードバックを入力します。

ソースリポジトリの問題のトラブルシューティング

以下の情報は、のソースリポジトリに関する一般的な問題のトラブルシューティングに役立ちます。CodeCatalyst

トピック

- スペースの最大ストレージ容量に達し、警告またはエラーが表示されます。
- Amazon CodeCatalyst ソースリポジトリを複製またはプッシュしようとするとエラーが表示されます
- Amazon CodeCatalyst ソースリポジトリにコミットまたはプッシュしようとするとエラーが表示されます
- 自分のプロジェクトにはソースリポジトリが必要です。
- 私のソースリポジトリは新品ですが、コミットが含まれています。
- デフォルトブランチとして別のブランチにしたい
- プルリクエストのアクティビティに関するメールを受信しています。
- 個人アクセストークン (PAT) を忘れてしまいました。
- プルリクエストには、期待した変更が表示されません。
- プルリクエストのステータスは「マージ不可」と表示されます。

スペースの最大ストレージ容量に達し、警告またはエラーが表示されます。

問題:の 1 つ以上のソースリポジトリにコードをコミットしたいのですが CodeCatalyst、エラーが表示されます。コンソールのソースリポジトリページに、スペースのストレージ制限に達したというメッセージが表示されます。

解決方法:プロジェクトまたはスペースでの役割に応じて、1 つ以上のソースリポジトリのサイズを小さくしたり、未使用のソースリポジトリを削除したり、請求階層をストレージの多いものに変更したりできます。

- プロジェクト内のソースリポジトリのサイズを小さくするには、未使用のブランチを削除できます。詳細については、「ブランチの削除」および「寄稿者ロール」を参照してください。
- スペース全体のストレージを減らすために、未使用のソースリポジトリを削除できます。詳細については、「ソースリポジトリの削除」および「プロジェクト管理者ロール」を参照してください。
- スペースに利用できるストレージの量を増やすには、請求レベルをストレージの多いものに変更できます。詳細については、『Amazon CodeCatalyst 管理者ガイド』の「 CodeCatalyst 請求階層の変更」を参照してください。

Amazon CodeCatalyst ソースリポジトリを複製またはプッシュしようとするとエラーが表示されます

問題:ソースリポジトリをローカルコンピュータまたは統合開発環境 (IDE) に複製しようとすると、権限エラーが表示されます。

解決方法: AWS Builder ID 用の個人アクセストークン (PAT) がない、PAT を使用して認証情報管理システムを設定していない、PAT の有効期限が切れている可能性があります。次の 1 つまたは複数の解決策を試してください。

- 個人アクセストークン (PAT) を作成します。詳細については、「個人用アクセストークンを使用してリポジトリアクセスをユーザーに付与する」を参照してください。
- ソースリポジトリを含むプロジェクトへの招待を受け入れ、自分がまだそのプロジェクトのメンバーであることを確認してください。ソースリポジトリは、そのプロジェクトのアクティブなメンバーでないとクローンできません。コンソールにサインインし、ソースリポジトリを複製しようとしているスペースとプロジェクトに移動してみます。スペースのプロジェクトリストにそのプロジェクトが表示されない場合は、そのプロジェクトのメンバーではないか、そのプロジェクトへの

招待を受けていないかのどちらかです。詳細については、「招待を受け入れて Builder ID AWS を作成する」を参照してください。

- クローンコマンドが正しい形式で、 AWS ビルダー ID が含まれていることを確認してください。例:

```
https://LiJuan@git.us-west-2.codecatalyst.aws/
v1/ExampleCorp/MyExampleProject/MyExampleRepo
```

- AWS CLI を使用して、 AWS Builder ID に PAT が関連付けられていることと、その有効期限が切れていないことを確認します。PAT がない場合や PAT の有効期限が切れている場合は、作成してください。詳細については、「個人用アクセストークンを使用してリポジトリアクセスをユーザーに付与する」を参照してください。
- コードをローカルリポジトリや IDE に複製するのではなく、ソースリポジトリ内のコードを操作する開発環境を作成してみてください。詳細については、「開発環境の作成」を参照してください。

Amazon CodeCatalyst ソースリポジトリにコミットまたはプッシュしようとするとエラーが表示されます

問題:ソースリポジトリにプッシュしようとすると、アクセス権限エラーが表示されます。

考えられる解決方法:コードの変更をコミットしたりプロジェクトにプッシュしたりできるロールがプロジェクトにない可能性があります。ソースリポジトリに変更をプッシュしようとしているプロジェクトにおける自分の役割を確認してください。詳細については、「メンバーとそのプロジェクトロールのリストの取得」および「ユーザーロールによるアクセス許可の付与」を参照してください。

変更のコミットとプッシュを許可するロールがある場合、変更をコミットしようとしているブランチに、コード変更をそのブランチにプッシュできないようにするブランチルールが設定されている可能性があります。代わりに、ブランチを作成してコードをそのブランチにプッシュしてみてください。詳細については、「ブランチルールを使用してブランチに対して許可されるアクションを管理する」を参照してください。

自分のプロジェクトにはソースリポジトリが必要です。

問題:プロジェクトにソースリポジトリがないか、プロジェクト用に別のソースリポジトリが必要です。

考えられる解決策:一部のプロジェクトはリソースなしで作成されています。プロジェクトのメンバーであれば、 CodeCatalystそのプロジェクトのソースリポジトリをで作成できます。GitHub GitHub スペース管理者ロールを持つユーザーがリポジトリをインストールしてアカウントに接続した場合、プロジェクト管理者ロールを持っていれば、 GitHub 使用可能なリポジトリにリンクしてプロジェクトに追加できます。詳細については、「ソースリポジトリの作成」と「ソースリポジトリのリンク」を参照してください。

私のソースリポジトリは新品ですが、コミットが含まれています。

問題:ソースリポジトリを作成したばかりです。空のはずなのに、コミット、ブランチ、README.mdファイルが入っています。

考えられる修正:これは想定どおりの動作です。 CodeCatalyst のすべてのソースリポジトリには、デフォルトブランチをサンプルコード (サンプルコードを含むブループリントを使用してプロジェクト用に作成した場合) またはリポジトリの README ファイル用のテンプレートマークダウンファイルのいずれかを設定する初期コミットが含まれます。mainコンソールと Git クライアントで追加のブランチを作成できます。コンソールでファイルを作成および編集したり、開発環境と Git クライアントでファイルを削除したりできます。

デフォルトブランチとして別のブランチにしたい

問題:ソースリポジトリにはという名前のデフォルトブランチが付いていますがmain、デフォルトブランチとして別のブランチを使いたいです。

考えられる解決方法:のソースリポジトリのデフォルトブランチを変更または削除することはできません。 CodeCatalyst追加のブランチを作成して、そのブランチをワークフローのソースアクションで使用できます。 GitHub リポジトリをリンクして、プロジェクトのリポジトリとして使用することもできます。

プルリクエストのアクティビティに関するメールを受信しています。

問題:プルリクエストアクティビティに関するメール通知をサインアップまたは設定していないのに、それでも届いてしまう。

考えられる解決方法:プルリクエストアクティビティに関するメール通知が自動的に送信されます。詳細については、「Amazon でのプルリクエストによるコードの確認 CodeCatalyst」を参照してください。

個人アクセストークン (PAT) を忘れてしまいました。

問題:ソースリポジトリのコードのクローニング、プッシュ、プルに PAT を使用してきましたが、トークンの値がなくなり、コンソールにも見つかりません。 CodeCatalyst

解決方法:この問題を解決する最も簡単な方法は、別の PAT を作成し、その新しい PAT を使用するように認証情報マネージャまたは IDE を設定することです。PAT の値は、作成時にのみ表示されます。この値を失うと、元に戻すことはできません。詳細については、「個人用アクセストークンを使用してリポジトリアクセスをユーザーに付与する」を参照してください。

プルリクエストには、期待した変更が表示されません。

問題:プルリクエストを作成したのに、ソースブランチとターゲットブランチ間で期待していた変更が表示されません。

考えられる解決方法:これはいくつかの問題が原因と考えられます。次の 1 つまたは複数の解決策を試してください。

- 古いリビジョン間の変更を確認している場合もあれば、最新の変更を確認していない場合もあります。ブラウザを更新して、表示したいリビジョン間の比較を選択していることを確認してください。
- プルリクエストのすべての変更をコンソールに表示できるわけではありません。たとえば、Git サブモジュールはコンソールに表示できないため、プルリクエストのサブモジュールの違いを確認することはできません。相違点の中には大きすぎて表示できないものもあります。詳細については、「のソースリポジトリのクォータ CodeCatalyst」および「ファイルの表示」を参照してください。
- プルリクエストには、マージベースと選択したリビジョンとの違いが表示されます。プルリクエストを作成すると、ソースブランチの先端と宛先ブランチの先端の違いが表示されます。プルリクエストが作成されると、リビジョンとマージベースの違いが表示されます。マージベースは、リビジョンが作成された時点でターゲットブランチの先端にあったコミットです。マージベースはリビジョンごとに変わる可能性があります。Git の違いとマージベースの詳細については、Git ドキュメントのを参照してくださいgit-merge-base。

プルリクエストのステータスは「マージ不可」と表示されます。

問題:プルリクエストをマージしたいが、ステータスが「マージ不可」と表示される。

考えられる解決方法:これは 1 つ以上の問題が原因の可能性があります。

- プルリクエストをマージする前に、プルリクエストに必要なすべてのレビュアーがプルリクエストを承認する必要があります。名前の横に時計のアイコンが付いているレビュアーがあれば、必要なレビュアーのリストを確認してください。時計アイコンは、レビュアーがプルリクエストを承認していないことを示します。

  > **Note**
  >
  > プルリクエストを承認する前に必須のレビュアーがプロジェクトから削除されている場合、プルリクエストをマージすることはできません。プルリクエストを閉じて、新しいプルリクエストを作成してください。

- ソースブランチとターゲットブランチ間でマージコンフリクトが発生する可能性があります。CodeCatalyst は、考えられるすべての Git マージ戦略とオプションをサポートしているわけではありません。Dev Environment でブランチのマージコンフリクトを評価したり、リポジトリをクローンして IDE や Git ツールを使用してマージコンフリクトを見つけて解決したりできます。詳細については、「プルリクエストのマージ」を参照してください。

プロジェクトとブループリントのトラブルシューティング

このセクションは、Amazon でプロジェクトや設計図を操作する際に発生する可能性のある一般的な問題のトラブルシューティングに役立ちます。 CodeCatalyst

AWS Fargateアパッチ・メイヴン-3.8.6のブループリントの依存関係が欠落しているJava API

問題:AWS Fargateブループリントで Java API から作成されたプロジェクトでは、依存関係がないというエラーでワークフローが失敗します。apache-maven-3.8.6ワークフローが失敗し、次の例のような出力が表示されます。

```
Step 8/25 : RUN wget https://dlcdn.apache.org/maven/maven-3/3.8.6/binaries/apache-
maven-3.8.6-bin.tar.gz -P /tmp
---> Running in 1851ce6f4d1b
[91m--2023-03-10 01:24:55--  https://dlcdn.apache.org/maven/maven-3/3.8.6/binaries/
apache-maven-3.8.6-bin.tar.gz
[0m[91mResolving dlcdn.apache.org (dlcdn.apache.org)...
[0m[91m151.101.2.132, 2a04:4e42::644
Connecting to dlcdn.apache.org (dlcdn.apache.org)|151.101.2.132|:443...
[0m[91mconnected.
```

```
[0m[91mHTTP request sent, awaiting response... [0m[91m404 Not Found
2023-03-10 01:24:55 ERROR 404: Not Found.
[0mThe command '/bin/sh -c wget https://dlcdn.apache.org/maven/maven-3/3.8.6/binaries/
apache-maven-3.8.6-bin.tar.gz -P /tmp' returned a non-zero code: 8
[Container] 2023/03/10 01:24:55 Command failed with exit status 8
```

解決策:以下の手順を使用してブループリント Dockerfile を更新します。

1. 検索バーに「」apache-maven-3.8.6 と入力して、ブループリント付き Java API で作成したプロジェクト内の Dockerfile を検索します。AWS Fargate
2. Dockerfile (/static-assets/app/Dockerfile) maven:3.9.0-amazoncorretto-11 をベースイメージとして使用するように更新し、パッケージへの依存関係を削除します。apache-maven-3.8.6
3. (推奨) Maven ヒープサイズを 6 GB に更新することもお勧めします。

以下は Docker ファイルの例です。

```
FROM maven:3.9.0-amazoncorretto-11 AS builder

COPY ./pom.xml ./pom.xml
COPY src ./src/

ENV MAVEN_OPTS='-Xmx6g'

RUN mvn -Dmaven.test.skip=true clean package

FROM amazoncorretto:11-alpine

COPY –from=builder target/CustomerService-0.0.1.jar CustomerService-0.0.1.jar
EXPOSE 80
CMD ["java","-jar","-Dspring.profiles.active=prod","/CustomerService-0.0.1.jar", "–
server.port=80"]
```

OnPullRequest最新の3層ウェブアプリケーションブループリントワークフローがAmazonの権限エラーで失敗する CodeGuru

問題:プロジェクトのワークフローを実行しようとすると、次のメッセージが表示されてワークフローが実行されません。

```
Failed at codeguru_codereview: The action failed during runtime. View the action's logs
 for more details.
```

解決策:このアクションが失敗する原因の 1 つは、IAM ロールポリシーに権限がないことが原因と考えられます。 CodeCatalyst 接続側で使用されているサービスロールのバージョンに、codeguru_codereview AWS アカウント アクションを正常に実行するために必要な権限がないことが原因です。この問題を解決するには、サービスロールを必要な権限で更新するか、ワークフローに使用されるサービスロールを Amazon CodeGuru と Amazon CodeGuru Reviewer に必要な権限を持つサービスロールに変更する必要があります。ワークフローを正常に実行できるように、次の手順を使用してロールを検索し、ロールポリシーのアクセス権限を更新します。

> **Note**
>
> これらの手順は、の以下のワークフローに適用されます CodeCatalyst。
>
> - モダン 3 層 Web OnPullRequestアプリケーションブループリントで作成されたプロジェクトに用意されているワークフロー。 CodeCatalyst
> - Amazon CodeGuru または Amazon CodeGuru Reviewer CodeCatalyst にアクセスするアクションでプロジェクトに追加されたワークフロー。

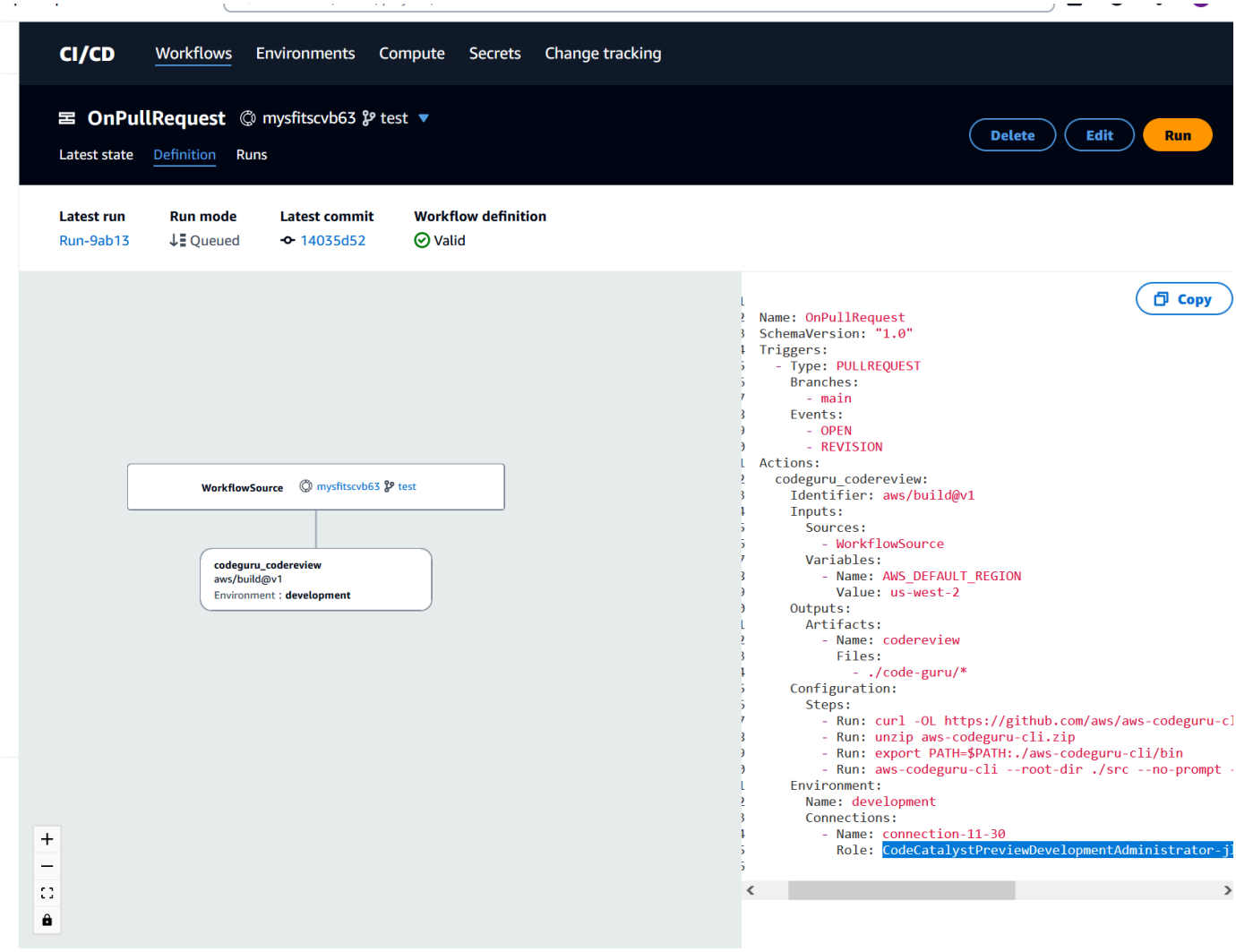
各プロジェクトには、AWS アカウントでプロジェクトに接続されているから提供されるロールと環境を使用するアクションを含むワークフローが含まれています。 CodeCatalystアクションと指定されたポリシーを含むワークフローは、ソースリポジトリの /.codecatalyst/workflows ディレクトリに保存されます。既存のワークフローに新しいロール ID を追加しない限り、ワークフロー YAML を変更する必要はありません。YAML テンプレート要素とフォーマットについては、を参照してください。ワークフロー YAML 定義

ロールポリシーを編集し、ワークフロー YAML を検証するための大まかな手順は次のとおりです。

ワークフロー YAML でロール名を参照してポリシーを更新するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. CodeCatalyst 自分のスペースに移動します。プロジェクトに移動します。
3. [CI/CD] を選択し、[ワークフロー] を選択します。
4. というタイトルの付いたワークフローを選択します。OnPullRequest[Definition] (定義) タブを選択します。

5. ワークフロー YAML の codeguru_codereview **Role:** アクションの下のフィールドに、ロール名を書き留めておきます。これは IAM で変更するポリシーを含むロールです。次の例はロール名を示しています。



6. 次のいずれかを実行します。

   - (推奨) プロジェクトに接続されているサービスロールを、Amazon CodeGuru と Amazon CodeGuru Reviewer に必要な権限で更新します。CodeCatalystWorkflowDevelopmentRole-*spaceName*ロールには一意の識別子が付加された名前が付きます。ロールとロールポリシーの詳細については、を参照してくださいCodeCatalystWorkflowDevelopmentRole-***spaceName*** サービスロールについて。次のステップに進み、IAM のポリシーを更新します。

> **Note**
>
> ロールとポリシーを使用して、AWSAWS アカウントへの管理者アクセス権が必要です。

- ワークフローに使用されるサービスロールを Amazon CodeGuru と Amazon CodeGuru Reviewer に必要な権限を持つサービスロールに変更するか、必要な権限を持つ新しいロールを作成します。

7. AWS Management Console にサインインして、IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。

   IAM コンソールで、ステップ 5 のロール (など) を探します。`CodeCatalystPreviewDevelopmentRole`

8. ステップ 5 のロールで、`codeguru-reviewer:*codeguru:*`およびのアクセス権限を含むようにアクセス権限ポリシーを変更します。これらのアクセス権限を追加すると、アクセス権限ポリシーは次のようになるはずです。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "cloudformation:*",
                "lambda:*",
                "apigateway:*",
                "ecr:*",
                "ecs:*",
                "ssm:*",
                "codedeploy:*",
                "s3:*",
                "iam:DeleteRole",
                "iam:UpdateRole",
                "iam:Get*",
                "iam:TagRole",
                "iam:PassRole",
                "iam:CreateRole",
                "iam:AttachRolePolicy",
                "iam:DetachRolePolicy",
                "iam:PutRolePolicy",
                "iam:CreatePolicy",
```

```
                "iam:DeletePolicy",
                "iam:CreatePolicyVersion",
                "iam:DeletePolicyVersion",
                "iam:PutRolePermissionsBoundary",
                "iam:DeleteRolePermissionsBoundary",
                "sts:AssumeRole",
                "elasticloadbalancing:DescribeTargetGroups",
                "elasticloadbalancing:DescribeListeners",
                "elasticloadbalancing:ModifyListener",
                "elasticloadbalancing:DescribeRules",
                "elasticloadbalancing:ModifyRule",
                "cloudwatch:DescribeAlarms",
                "sns:Publish",
                "sns:ListTopics",
                "codeguru-reviewer:*",
                "codeguru:*"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

9. ポリシーを修正したら、 CodeCatalyst に戻ってワークフローを再実行してください。

まだ問題を解決したいとお考えですか?

Amazon CodeCatalyst にアクセスするか、Support フィードバックフォームに記入してください。「リクエスト情報」セクションの「お手伝いできること」に、お客様が Amazon CodeCatalyst のお客様であることを明記してください。問題に最大限効率的に対処できるように、できるだけ詳しく説明してください。

ワークフローに関する問題のトラブルシューティング

Amazon のワークフローに関連する問題のトラブルシューティングについては、以下のセクションを参照してください CodeCatalyst。ワークフローの詳細については、「でワークフローを使用して構築、テスト、デプロイする CodeCatalyst」を参照してください。

トピック

- 「ワークフローは非アクティブです」というメッセージを修正するにはどうすればよいですか？

「ワークフローは非アクティブです」というメッセージを修正するにはどうすればよいですか？

問題： CodeCatalyst コンソールの CI/CD のワークフロー で、ワークフローに次のメッセージが表示されます。

```
Workflow is inactive.
```

このメッセージは、ワークフロー定義ファイルに、現在使用しているブランチに適用されないトリガーが含まれていることを示します。例えば、ワークフロー定義ファイルには、mainブランチを参照するPUSHトリガーが含まれている場合がありますが、特徴量ブランチを使用しているとします。機能ブランチで行った変更は には適用されずmain、 でワークフロー実行が開始されないためmain、 はブランチのワークフローを CodeCatalyst 廃止し、 としてマークしますInactive。

解決方法:

機能ブランチでワークフローを開始する場合は、次の操作を実行できます。

- 機能ブランチのワークフロー定義ファイルで、 Branches プロパティを Triggersセクションから削除して、次のようになります。

  ```
  Triggers:
    - Type: PUSH
  ```

  この設定により、特徴量ブランチを含む任意のブランチへのプッシュ時にトリガーがアクティブになります。トリガーがアクティブ化 CodeCatalyst されている場合、プッシュ先のブランチのワークフロー定義ファイルとソースファイルを使用してワークフロー実行を開始します。
- 機能ブランチのワークフロー定義ファイルで、 Triggersセクションを削除し、ワークフローを手動で実行します。
- 機能ブランチのワークフロー定義ファイルで、 PUSHセクションを変更してmain、別のブランチ (など) ではなく機能ブランチを参照するようにします。

> **Important**
>
> これらの変更をmainブランチにマージする予定がない場合は、これらの変更をコミットしないように注意してください。

ワークフロー定義ファイルの編集の詳細については、「」を参照してくださいワークフローの作成。

トリガーについての詳細は、「トリガーを使用したワークフローの自動実行の開始」を参照してください。

「ワークフロー定義に *n* エラーがある」エラーを修正するにはどうすればよいですか？

問題： 次のいずれかのエラーメッセージが表示されます。

エラー 1:

CI/CD のワークフローページでは、ワークフローの名前の下に次の内容が表示されます。

`Workflow definition has n errors`

エラー 2:

ワークフローの編集中に検証ボタンを選択すると、 CodeCatalyst コンソールの上部に次のメッセージが表示されます。

```
The workflow definition has errors. Fix the errors and choose Validate to
verify your changes.
```

エラー 3:

ワークフローの詳細ページに移動すると、ワークフロー定義フィールドに次のエラーが表示されます。

```
n errors
```

解決方法:

- CI/CD を選択し、ワークフロー を選択し、エラーのあるワークフローの名前を選択します。上部にあるワークフロー定義フィールドで、エラーへのリンクを選択します。エラーに関する詳細は、ページの下部に表示されます。エラーのトラブルシューティングのヒントに従って問題を解決します。
- ワークフロー定義ファイルが YAML ファイルであることを確認します。
- ワークフロー定義ファイルの YAML プロパティが適切なレベルでネストされていることを確認します。ワークフロー定義ファイルにプロパティをネストする方法を確認するには、「」を参照ワークフロー YAML 定義するか、「」からリンクされているアクションのドキュメントを参照してください CodeCatalyst ワークフローへのアクションの追加。
- アスタリスク (*) やその他の特殊文字が正しくエスケープされていることを確認してください。エスケープするには、一重引用符または二重引用符を追加します。例:

  ```
  Outputs:
    Artifacts:
      - Name: myartifact
        Files:
          - "**/*"
  ```

  ワークフロー定義ファイルの特殊文字の詳細については、「」を参照してください構文のガイドラインと規則。
- ワークフロー定義ファイルの YAML プロパティで、正しい大文字と小文字が使用されていることを確認します。大文字と小文字のルールの詳細については、「」を参照してください構文のガイドラインと規則。各プロパティの正しい大文字と小文字を確認するには、「」を参照ワークフロー

YAML 定義するか、「」から にリンクされているアクションのドキュメントを参照してください CodeCatalyst ワークフローへのアクションの追加。

- SchemaVersion プロパティが存在し、ワークフロー定義ファイルで正しいバージョンに設定されていることを確認します。詳細については、「SchemaVersion」を参照してください。
- ワークフロー定義ファイルの Triggersセクションに、必要なプロパティがすべて含まれていることを確認してください。必要なプロパティを確認するには、ビジュアルエディタでトリガーを選択し、情報が不足しているフィールドを検索するか、「」のトリガーリファレンスドキュメントを参照してくださいTriggers。
- ワークフロー定義ファイルの DependsOnプロパティが正しく設定されており、循環依存関係が導入されていないことを確認します。詳細については、「他のアクションに依存するようにアクションを設定する」を参照してください。
- ワークフロー定義ファイルの Actionsセクションに少なくとも 1 つのアクションが含まれていることを確認します。詳細については、「アクション」を参照してください。
- 各アクションに必要なプロパティがすべて含まれていることを確認してください。必要なプロパティを確認するには、ビジュアルエディタでアクションを選択し、情報が不足しているフィールドを検索するか、 からリンクされているアクションのドキュメントを参照してください CodeCatalyst ワークフローへのアクションの追加。
- すべての入力アーティファクトに対応する出力アーティファクトがあることを確認します。詳細については、「出力アーティファクトの定義」を参照してください。
- あるアクションで定義された変数がエクスポートされ、他のアクションで使用できるようにしてください。詳細については、「他のアクションで使用できるように変数をエクスポートする」を参照してください。

「認証情報が見つかりません」と「」のExpiredTokenエラーを修正するにはどうすればよいですか？

問題: の作業中にチュートリアル: Amazon EKS にアプリケーションをデプロイする、開発マシンのターミナルウィンドウに次のエラーメッセージの 1 つまたは両方が表示されます。

```
Unable to locate credentials. You can configure credentials by running "aws
configure".
```

```
ExpiredToken: The security token included in the request is expired
```

解決方法:

これらのエラーは、 AWS サービスへのアクセスに使用している認証情報の有効期限が切れていることを示しています。この場合、 `aws configure` コマンドを実行しないでください。代わりに、次の手順を使用して AWS アクセスキーとセッショントークンを更新します。

AWS アクセスキーとセッショントークンを更新するには

1. Amazon EKS チュートリアル () 全体を使用しているユーザーの AWS アクセスポータル URL、ユーザー名、パスワードがあることを確認しますcodecatalyst-eks-user。ステップ 1: 開発マシンをセットアップする チュートリアルの完了時に、これらの項目を設定しておく必要があります。

   > **Note**
   >
   > この情報がない場合は、IAM Identity Center codecatalyst-eks-userの詳細ページに移動し、パスワードのリセット、ワンタイムパスワードの生成 [...]、パスワードのリセットをもう一度選択して、画面に情報を表示します。

2. 次のいずれかを行います。

   - AWS アクセスポータル URL をブラウザのアドレスバーに貼り付けます。

     または

   - アクセス AWS ポータルページが既にロードされている場合は、更新します。

3. まだサインインしていない場合は、 codecatalyst-eks-userのユーザー名とパスワードでサインインします。
4. を選択しAWS アカウント、codecatalyst-eks-userユーザーとアクセス許可セット AWS アカウント を割り当てた の名前を選択します。
5. アクセス許可セット名 (codecatalyst-eks-permission-set) の横にあるコマンドラインまたはプログラムによるアクセスを選択します。
6. ページの中央にあるコマンドをコピーします。これらは次のようになります。

   ```
   export AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
   export AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
   export AWS_SESSION_TOKEN="session-token"
   ```

   セッション####は長いランダム文字列です。

7. コマンドを開発マシンのターミナルプロンプトに貼り付け、Enter キーを押します。

新しいキーとセッショントークンがロードされます。

これで、認証情報が更新されました。これで AWS CLI、、eksctl、および kubectl コマンドが機能します。

「サーバーに接続できません」というエラーを修正するにはどうすればよいですか？

問題：で説明されているチュートリアルを実行しているときにチュートリアル: Amazon EKS にアプリケーションをデプロイする、開発マシンのターミナルウィンドウに次のようなエラーメッセージが表示されます。

```
Unable to connect to the server: dial tcp: lookup long-string.gr7.us-west-2.eks.amazonaws.com on 1.2.3.4:5: no such host
```

解決方法:

このエラーは通常、kubectlユーティリティが Amazon EKS クラスターへの接続に使用している認証情報の有効期限が切れていることを示します。この問題を解決するには、ターミナルプロンプトで次のコマンドを入力して認証情報を更新します。

```
aws eks update-kubeconfig --name codecatalyst-eks-cluster --region us-west-2
```

コードの説明は以下のとおりです。

- *codecatalyst-eks-cluster* は、Amazon EKS クラスターの名前に置き換えられます。
- *us-west-2* は、クラスターがデプロイされている AWS リージョンに置き換えられます。

ビジュアルエディタに CodeDeploy フィールドがないのはなぜですか？

問題: Amazon ECS へのデプロイアクションを使用していて、ワークフローのビジュアルエディタCodeDeploy AppSpecに などの CodeDeploy フィールドが表示されません。この問題は、サービスフィールドで指定した Amazon ECS サービスがブルー/グリーンデプロイを実行するように設定されていないために発生する可能性があります。

解決方法:

- Amazon ECS へのデプロイアクションの設定タブで、別の Amazon ECS サービスを選択します。詳細については、「ワークフローを使用した Amazon Elastic Container Service (ECS) へのアプリケーションのデプロイ」を参照してください。
- ブルー/グリーンデプロイを実行するように選択した Amazon ECS サービスを設定します。ブルー/グリーンデプロイの設定の詳細については、「Amazon Elastic Container Service デベロッパーガイド」の「 を使用したブルー/グリーンデプロイ CodeDeploy」を参照してください。

IAM 機能エラーを修正するにはどうすればよいですか？

問題: AWS CloudFormation スタックのデプロイアクションを使用していて、スタックのデプロイ AWS CloudFormation アクションのログ##[error] requires capabilities: [*capability-name*]に が表示されます。

解決方法： ワークフロー定義ファイルに 機能を追加するには、以下の手順を実行します。IAM 機能の詳細については、「IAM ユーザーガイド」の AWS CloudFormation 「 テンプレートでの IAM リソースの承認」を参照してください。

Visual

ビジュアルエディタを使用して IAM 機能を追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. ビジュアル を選択します。
7. ワークフロー図で、スタックのデプロイ AWS CloudFormation アクションを選択します。
8. [設定] タブを選択します。
9. 下部で、詳細 - オプション を選択します。
10. 機能 ドロップダウンリストから、エラーメッセージに記載されている機能の横にあるチェックボックスをオンにします。機能がリストにない場合は、YAML エディタを使用して追加します。

11. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
12. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。
13. 新しいワークフロー実行が自動的に開始しない場合は、ワークフローを手動で実行して、変更によってエラーが修正されたかどうかを確認します。ワークフローの手動実行の詳細については、「」を参照してくださいワークフローの手動実行の開始。

YAML

YAML エディタを使用して IAM 機能を追加するには

1. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
2. プロジェクトを選択します。
3. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。
4. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。
5. [編集] を選択します。
6. YAML を選択します。
7. AWS CloudFormation スタックのデプロイアクションで、次のようなcapabilitiesプロパティを追加します。

```
DeployCloudFormationStack:
  Configuration:
    capabilities: capability-name
```

capability-name を、エラーメッセージに表示されている IAM 機能の名前に置き換えます。複数の機能を一覧表示するには、カンマを使用し、スペースは使用しません。詳細については、「」の「 capabilitiesプロパティの説明」を参照してくださいAWS CloudFormation 「スタックのデプロイ」アクション YAML 定義。

8. (オプション) 検証 を選択して、コミットする前にワークフローの YAML コードを検証します。
9. コミット を選択し、コミットメッセージを入力し、もう一度コミット を選択します。

10. 新しいワークフロー実行が自動的に開始しない場合は、ワークフローを手動で実行して、変更によってエラーが修正されたかどうかを確認します。ワークフローの手動実行の詳細については、「」を参照してくださいワークフローの手動実行の開始。

「npm install」エラーを修正するにはどうすればよいですか？

問題： AWS CDK デプロイアクションまたはAWS CDK ブートストラップアクションがnpm installエラーで失敗します。このエラーは、 アクションではアクセスできないプライベートノードパッケージマネージャー (npm) レジストリに AWS CDK アプリケーションの依存関係を保存しているため発生する可能性があります。

解決方法: 次の手順に従って、追加のレジストリと認証情報で AWS CDK アプリケーションの cdk.json ファイルを更新します。

開始する前に

1. 認証情報のシークレットを作成します。これらのシークレットは、クリアテキストの同等のものを提供するのではなく、 cdk.json ファイルで参照します。シークレットを作成するには：

   a. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。

   b. プロジェクトを選択します。

   c. ナビゲーションペインで CI/CD を選択し、シークレット を選択します。

   d. 次のプロパティを使用して 2 つのシークレットを作成します。

最初のシークレット	2 番目のシークレット
名前: npmUsername 値: *npm-username* 。*npm-username* はプライベート npm レジストリへの認証に使用されるユーザー名です。 (オプション) 説明: The username used to authenticate to the private npm registry.	名前: npmAuthToken 値： 。ここで*npm-auth-token* 、 *npm-auth-token* はプライベート npm レジストリへの認証に使用されるアクセストークンです。npm アクセストークンの詳細については、npm ドキュメントの「アクセストークンについて」を参照してください。

最初のシークレット	2 番目のシークレット
	(オプション) 説明: `The access token used to authenticate to the private npm registry.`

シークレットの詳細については、「」を参照してくださいワークフローでのシークレットの設定と使用。

2. シークレットを環境変数として AWS CDK アクションに追加します。アクションは、実行時に変数を実際の値に置き換えます。シークレットを追加するには :

   a. ナビゲーションペインで CI/CD を選択し、ワークフロー を選択します。

   b. ワークフローの名前を選択します。ワークフローが定義されているソースリポジトリまたはブランチ名でフィルタリングすることも、ワークフロー名でフィルタリングすることもできます。

   c. [編集] を選択します。

   d. ビジュアル を選択します。

   e. ワークフロー図で、 AWS CDK アクションを選択します。

   f. [入力] タブを選択します。

   g. 次のプロパティを持つ 2 つの変数を追加します。

最初の変数	2 番目の変数
名前: `NPMUSER` 値: `${Secrets.npmUsername}`	名前: `NPMTOKEN` 値: `${Secrets.npmAuthToken}`

これで、シークレットへの参照を含む 2 つの変数ができました。

ワークフロー定義ファイルの YAML コードは次のようになります。

> **Note**
>
> 次のコードサンプルはAWS CDK ブートストラップアクションからのものです。AWS CDK デプロイアクションは似ています。

```
Name: CDK_Bootstrap_Action
SchemaVersion: 1.0
Actions:
  CDKBootstrapAction:
    Identifier: aws/cdk-bootstrap@v1
    Inputs:
      Variables:
        - Name: NPMUSER
          Value: ${Secrets.npmUsername}
        - Name: NPMTOKEN
          Value: ${Secrets.npmAuthToken}
      Sources:
        - WorkflowSource
    Environment:
      Name: Dev2
      Connections:
        - Name: account-connection
          Role: codecatalystAdmin
    Configuration:
      Parameters:
        Region: "us-east-2"
```

これで、 cdk.json ファイル内の 変数NPMUSERと NPMTOKEN変数を使用する準備ができました。次の手順に進みます。

cdk.json ファイルを更新するには

1. AWS CDK プロジェクトのルートディレクトリに移動し、 cdk.json ファイルを開きます。
2. "app": プロパティを検索し、#####で示されているコードを含めるように変更します。

> **Note**
>
> 次のサンプルコードは TypeScript プロジェクトからのものです。 JavaScript プロジェクトを使用している場合、コードは同じではありませんが、似ています。

```
{
  "app": "npm set registry=https://your-registry/folder/CDK-package/ --
userconfig .npmrc && npm set //your-registry/folder/CDK-package/:always-auth=true
 --userconfig .npmrc && npm set //your-registry/folder/CDK-package/:_authToken=
\"${NPMUSER}\":\"${NPMTOKEN}\" && npm install  && npx ts-node --prefer-ts-exts bin/
hello-cdk.ts|js",
  "watch": {
    "include": [
      "**"
    ],
    "exclude": [
      "README.md",
      "cdk*.json",
      "**/*.d.ts",
      "**/*.js",
      "tsconfig.json",
      "package*.json",
...
```

3. 赤### で強調表示されているコードで、以下を置き換えます。

   - *your-registry/folder/CDK-package/* と、プライベートレジストリ内の AWS CDK プロジェクトの依存関係へのパス。
   - *hello-cdk.ts|.js* とエントリポイントファイルの名前。これは、使用している言語に応じて .ts (TypeScript) または .js (JavaScript) ファイルになります。

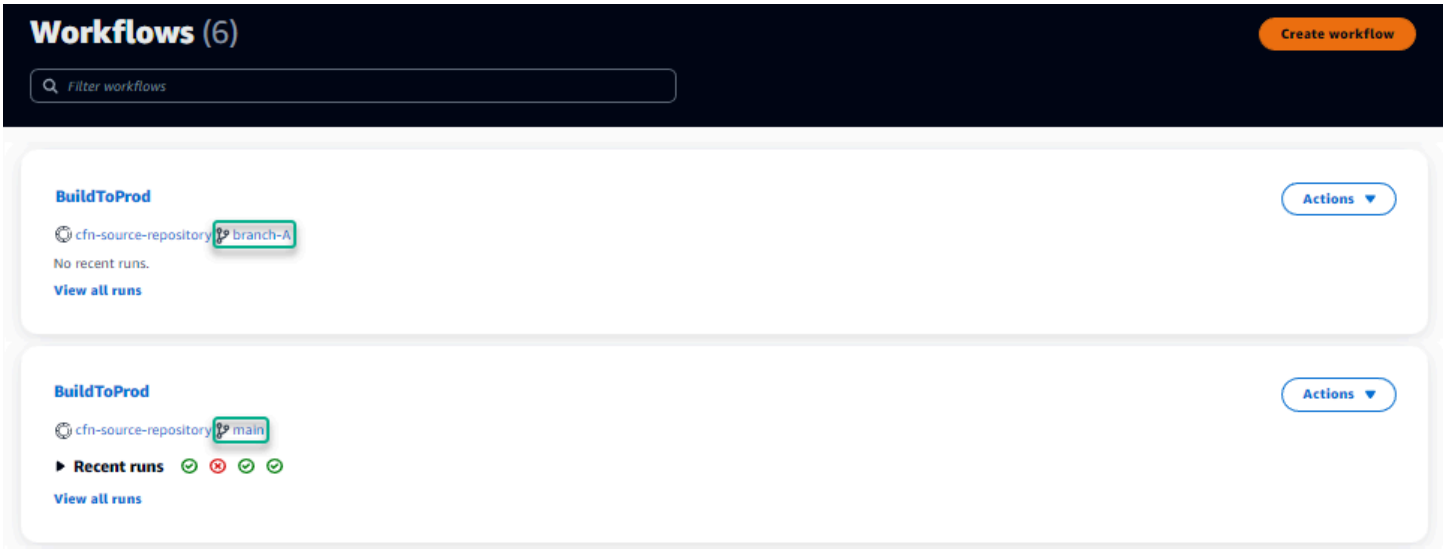
   > **Note**
   >
   > アクションは、*NPMUSER* 変数と *NPMTOKEN* 変数を、Secrets で指定した npm ユーザー名とアクセストークンに置き換えます。

4. cdk.json ファイルを保存します。

5. アクションを手動で再実行して、変更によってエラーが修正されたかどうかを確認します。アクションの手動実行の詳細については、「」を参照してくださいワークフローの手動実行の開始。

複数のワークフローが同じ名前を持つのはなぜですか？

ワークフローは、リポジトリごとにブランチごとに保存されます。2 つの異なるワークフローは、異なるブランチに存在する場合、同じ名前を持つことができます。ワークフローページで、ブランチ名を確認することで、同じ名前のワークフローを区別できます。詳細については、「ソースコードを整理して Amazon のブランチを操作する CodeCatalyst」を参照してください。



ワークフロー定義ファイルを別のフォルダに保存できますか？

いいえ、すべてのワークフロー定義ファイルを .codecatalyst/workflowsフォルダに保存する必要があります。複数の論理プロジェクトでモノリポジトリを使用している場合は、すべてのワークフロー定義ファイルを .codecatalyst/workflowsフォルダに配置し、トリガー内で FilesChangedプロパティを使用して、指定されたプロジェクトパスでワークフローをトリガーします。詳細については、「トリガーを使用したワークフローの自動実行の開始」を参照してください。

ワークフローにアクションを順番に追加するにはどうすればよいですか？

デフォルトでは、ワークフローにアクションを追加すると、依存関係はなく、他のアクションと並行して実行されます。

アクションを順番に配置したい場合は、 DependsOnフィールドを設定することで、別のアクションへの依存関係を設定できます。他のアクションの出力であるアーティファクトまたは変数を使用するようにアクションを設定することもできます。詳細については、「他のアクションに依存するようにアクションを設定する」を参照してください。

ワークフローが正常に検証されても実行時に失敗するのはなぜですか？

Validate ボタンを使用してワークフローを検証したが、ワークフローが失敗した場合は、検証機能の制限が原因である可能性があります。

ワークフロー設定のシークレット、環境、フリートなどの CodeCatalyst リソースを参照するエラーは、コミット中に登録されません。無効な参照が使用されている場合、エラーはワークフローの実行時にのみ識別されます。同様に、必須フィールドの欠落やアクション属性の誤字など、アクション設定にエラーがある場合、ワークフローの実行時にのみ識別されます。詳細については、「ワークフローの作成」を参照してください。

自動検出では、アクションのレポートが検出されない

問題： テストを実行するアクションに自動検出を設定しましたが、 によってレポートが検出されません CodeCatalyst。

解決方法： これはいくつかの問題が原因である可能性があります。次のソリューションを 1 つ以上試してください。

- テストの実行に使用されるツールが、 が CodeCatalyst 理解する形式のいずれかで出力を生成していることを確認します。例えば、 pytestがテストとコードのカバレッジレポートを検出 CodeCatalyst できるようにする場合は、次の引数を含めます。

  ```
  --junitxml=test_results.xml --cov-report xml:test_coverage.xml
  ```

  詳細については、「品質レポートタイプ」を参照してください。
- 出力のファイル拡張子が、選択した形式と一致していることを確認します。例えば、 を設定pytestして JUnitXML 形式で結果を生成する場合は、ファイル拡張子が であることを確認します.xml。詳細については、「品質レポートタイプ」を参照してください。
- 特定のフォルダを意図的に除外しない限りIncludePaths、 がファイルシステム全体 (**/*) を含むように設定されていることを確認します。同様に、レポートが配置されていることが予想されるディレクトリを除外ExcludePathsしないようにしてください。

- 特定の出力ファイルを使用するようにレポートを手動で設定した場合、自動検出から除外されます。詳細については、「品質レポート YAML の例」を参照してください。
- 出力が生成される前にアクションが失敗したため、自動検出でレポートが見つからない場合があります。例えば、ユニットテストが実行される前にビルドが失敗した可能性があります。

成功基準を設定した後、自動検出されたレポートでアクションが失敗する

問題： 自動検出を有効にして成功基準を設定すると、一部のレポートが成功基準を満たさず、結果としてアクションが失敗します。

解決方法： この問題を解決するには、次の解決策を 1 つ以上試してください。

- IncludePaths または を変更ExcludePathsして、関心のないレポートを除外します。
- 成功基準を更新して、すべてのレポートが合格できるようにします。例えば、2 つのレポートが 50% のラインカバレッジと 70% の別のレポートで見つかった場合は、最小ラインカバレッジを 50% に調整します。詳細については、「成功基準」を参照してください。
- 失敗したレポートを手動で設定したレポートに変換します。これにより、その特定のレポートに対して異なる成功基準を設定できます。詳細については、「レポートの成功基準の設定」を参照してください。

自動検出では不要なレポートが生成されます

問題： 自動検出を有効にすると、不要なレポートが生成されます。例えば、 は、 に保存されているアプリケーションの依存関係に含まれるファイルのコードカバレッジレポート CodeCatalyst を生成しますnode_modules。

解決方法： 不要なファイルを除外するようにExcludePaths設定を調整できます。例えば、 を除外するにはnode_modules、 を追加しますnode_modules/**/*。詳細については、「パスの包含/除外」を参照してください。

自動検出は、1 つのテストフレームワークに対して多数の小さなレポートを生成します。

問題： 特定のテストおよびコードカバレッジレポートフレームワークを使用すると、自動検出によって多数のレポートが生成されることに気付きました。例えば、Maven の Surefire プラグインを使用する場合、自動検出はテストクラスごとに異なるレポートを生成します。

解決方法： フレームワークは出力を 1 つのファイルに集約できる場合があります。例えば、Maven の Surefire プラグインを使用している場合は、 `npx junit-merge`を使用してファイルを手動で集計できます。完全な式は次のようになります。

```
mvn test; cd test-package-path/surefire-reports && npx junit-merge -d ./ && rm
 *Test.xml
```

CI/CD の下にリストされているワークフローがソースリポジトリのワークフローと一致しません

問題： CI/CD のワークフローページに表示されるワークフローが、ソースリポジトリの `~/.codecatalyst/workflows/`フォルダにあるワークフローと一致しません。次の不一致が表示される場合があります。

- ワークフローがワークフローページに表示されますが、対応するワークフロー定義ファイルがソースリポジトリに存在しません。
- ワークフロー定義ファイルはソースリポジトリに存在しますが、対応するワークフローはワークフローページに表示されません。
- ワークフローはソースリポジトリとワークフローページの両方に存在しますが、2 つは異なります。

この問題は、ワークフローページが更新される時間がない場合、またはワークフロークォータを超えた場合に発生する可能性があります。

解決方法:

- 待ちます。通常、ソースへのコミットから 2 ~ 3 秒待ってから、ワークフローページで変更を確認する必要があります。
- ワークフロークォータを超えた場合は、次のいずれかを実行します。

> **Note**
>
> ワークフロークォータを超えたかどうかを判断するには、「」を確認しワークフローのクォータ、文書化されたクォータをソースリポジトリまたはワークフローページのワークフローと照合します。クォータを超過したことを示すエラーメッセージがないため、自分で調査する必要があります。

- スペースクォータあたりのワークフローの最大数を超えた場合は、一部のワークフローを削除してから、ワークフロー定義ファイルに対してテストコミットを実行します。テストコミットの例は、ファイルにスペースを追加する場合です。
- 最大ワークフロー定義ファイルサイズのクォータを超えた場合は、ワークフロー定義ファイルを変更して長さを短くします。
- 1 つのソースイベントクォータで処理されるワークフローファイルの最大数を超えた場合は、複数のテストコミットを実行します。各コミットのワークフローの最大数よりも少ない数を変更します。
- 有料利用枠の請求をオンにして、ワークフローのクォータを増やします。詳細については、「Amazon CodeCatalyst 管理者ガイド」の「請求の管理」を参照してください。

ワークフローを作成または更新できない

問題： ワークフローを作成または更新したいが、変更をコミットしようとするとエラーが表示される。

解決方法： プロジェクトまたはスペース内のロールによっては、プロジェクト内のソースリポジトリにコードをプッシュするアクセス許可がない場合があります。ワークフローの YAML ファイルはリポジトリに保存されます。詳細については、「ワークフロー定義ファイル」を参照してください。Space 管理者ロール、プロジェクト管理者ロール、および寄稿者ロールにはすべて、プロジェクト内のリポジトリにコードをコミットしてプッシュするアクセス許可があります。

Contributor ロールはあるが、特定のブランチでワークフロー YAML への変更を作成またはコミットできない場合、そのロールを持つユーザーがその特定のブランチにコードをプッシュできないようにするブランチルールが、そのブランチに設定されている可能性があります。別のブランチでワークフローを作成するか、変更を別のブランチにコミットしてみてください。詳細については、「ブランチルールを使用してブランチに対して許可されるアクションを管理する」を参照してください。

検索に関する問題のトラブルシューティング CodeCatalyst

以下のセクションを参照して、検索に関する問題のトラブルシューティングを行ってください。CodeCatalystワークフローの詳細については、「でコード、問題、プロジェクト、ユーザーを検索する CodeCatalyst」を参照してください。

トピック

自分のプロジェクトでユーザーが見つからない

問題:ユーザーの詳細を表示しようとしても、そのユーザーの情報がプロジェクトに表示されません。

考えられる解決策:現在、検索ではプロジェクト内のユーザーの検索はサポートされていません。スペースにアクセスできるユーザーを検索するには、「This space in」に切り替えるか QuickSearch、高度なクエリ言語を使用して指定したプロジェクトフィルターをすべて削除してください。

自分のプロジェクトやスペースで探しているものが見当たらない

問題:特定の情報を検索しようとしても、結果が表示されません。

解決方法:コンテンツが更新されて検索結果に反映されるまでに数秒かかることがあります。大規模な更新には数分かかることがあります。

最近更新されていないリソースについては、検索を絞り込む必要があるかもしれません。キーワードをさらに追加するか、高度なクエリ言語を使用して絞り込むことができます。クエリの絞り込みについて詳しくは、を参照してください。検索クエリの改良

ページ間を移動すると、検索結果の数が変わり続ける

問題:次のページに移動すると検索結果の数が変わっているように見えるので、合計でいくつの結果があるかが分かりません。

解決方法:検索結果のページをナビゲートしているときに、クエリに一致する検索結果の数が変化することがあります。検索結果の数は、ページ間を移動するにつれて、より正確な一致件数を反映して更新されることがあります。

結果間を移動すると、「「test」の結果はありません」というメッセージが表示される場合があります。残りの結果にアクセスできない場合は、メッセージが表示されます。

検索クエリが完了していません

問題:検索クエリの結果が表示されず、時間がかかりすぎているようです。

解決方法:プログラムまたはチームの作業量が多いため、スペース内で同時に多数の検索が行われると、検索が完了しないことがあります。プログラムによる検索を実行している場合は、検索を一時停止するか減らしてください。それ以外の場合は、数秒後にもう一度試してください。

スペースに関連付けられたアカウントに関する問題のトラブルシューティング

では CodeCatalyst、 AWS アカウントをスペースに追加して、リソースにアクセス許可を付与し、請求の目的で使用できます。以下の情報は、 の関連アカウントに関する一般的な問題のトラブルシューティングに役立ちます CodeCatalyst。

トピック

- AWS アカウント 接続リクエストに無効なトークンエラーが表示される
- Amazon CodeCatalyst プロジェクトのワークフローが、設定されたアカウント、環境、または IAM ロールのエラーで失敗する
- プロジェクトを作成するには、関連付けられたアカウント、ロール、環境が必要です
- の Amazon CodeCatalyst Spaces ページにアクセスできない AWS Management Console
- 請求アカウントとは異なるアカウントが必要

AWS アカウント 接続リクエストに無効なトークンエラーが表示される

問題： 接続トークンを使用して接続リクエストを作成する場合、ページはトークンを受け入れず、トークンが無効であることを示すエラーが表示されます。

解決方法： スペースに追加するアカウント ID を必ず指定してください。アカウントを追加するには、 の管理権限を持っているAWS アカウントか、 管理者と連携できる必要があります。

アカウントを検証すると、 で新しいブラウザウィンドウが開きますAWS Management Console。コンソール側でログインするには、同じアカウントが必要です。以下を確認したら、もう一度試してください。

- スペースに追加するのと同じ AWS Management Consoleで AWS アカウント にログインしている。
- 米国西部 (オレゴン) AWS リージョン (`us-west-2`) を選択したAWS Management Console状態でにログインしている。

- 請求ページから到達し、スペースの指定された請求アカウントAWS アカウントとして を追加する場合は、そのアカウントがまだ別のスペースの請求アカウントではないことを確認してください。

Amazon CodeCatalyst プロジェクトのワークフローが、設定されたアカウント、環境、または IAM ロールのエラーで失敗する

問題： ワークフローが実行され、スペースに関連付けられた設定済みのアカウントまたは IAM ロールが見つからない場合は、ワークフロー YAML でロール、接続、および環境フィールドを手動で入力する必要があります。失敗したワークフローアクションを表示し、エラーメッセージが次のようになっているかどうかを確認します。

- ロールは、環境に関連付けられた接続では使用できません。
- アクションは成功しませんでした。ステータス: 失敗。アカウントの接続または環境に指定された値が無効です。接続がスペースに関連付けられ、環境がプロジェクトに関連付けられていることを確認します。
- アクションは成功しませんでした。ステータス: 失敗。IAM ロールに指定された値が無効です。名前が存在し、IAM ロールがアカウント接続に追加され、接続が既に Amazon CodeCatalyst スペースに関連付けられていることを確認します。

解決方法： ワークフローの YAML フィールドに、環境 、接続 、ロール の正確な値があることを確認します。環境を必要とする CodeCatalyst ワークフローアクションは、 AWSリソースを実行するか、AWSリソーススタックを生成するビルドまたはデプロイアクションです。

失敗したワークフローアクションブロックを選択し、ビジュアル を選択します。[設定] タブを選択します。環境 、接続名 、およびロール名 フィールドに値が入力されていない場合は、ワークフローを手動で更新する必要があります。ワークフロー YAML を編集するには、次の手順に従います。

- `/.codecatalyst` ディレクトリを展開し、 `/workflows` ディレクトリを展開します。ワークフロー YAML ファイルを開きます。IAM ロールとアカウント情報が、ワークフロー用に設定した YAML で指定されていることを確認します。例：

```
Actions:
  cdk_bootstrap:
    Identifier: action-@v1
    Inputs:
      Sources:
        - WorkflowSource
```

```
    Environment:
      Name: Staging
      Connections:
        - Name: account-connection
          Role: build-role
```

環境、接続、ロールの各プロパティは、 AWSリソースを使用して CodeCatalyst ワークフローのビルドおよびデプロイアクションを実行するために必要です。例については、「環境」、「接続」、「ロール」の CodeCatalyst 「ビルドアクションリファレンス YAML パラメータ」を参照してください。 ???

- スペースにアカウントが追加されていること、およびアカウントに適切な IAM ロールが追加されていることを確認します。Space 管理者ロールがある場合は、アカウントを調整または追加できます。詳細については、「接続された AWS リソースへのアクセスを許可する AWS アカウント」を参照してください。

プロジェクトを作成するには、関連付けられたアカウント、ロール、環境が必要です

問題： プロジェクト作成オプションで、プロジェクトにスペースに使用可能なアカウントが追加されていないか、プロジェクトが使用できるようにスペースに別のアカウントを追加する必要があります。

解決方法： Space 管理者ロールがある場合は、スペースにプロジェクトAWS アカウントに追加する権限を追加できます。また、管理者権限AWS アカウントを持っているか、AWS管理者と連携できるも必要です。

アカウントとロールがプロジェクト作成画面で利用可能であることを確認するには、まずアカウントとロールを追加する必要があります。詳細については、「接続された AWS リソースへのアクセスを許可する AWS アカウント」を参照してください。

ロールポリシーと呼ばれるロールポリシーを使用してサービスCodeCatalystWorkflowDevelopmentRole-***spaceName***ロールを作成するオプションがあります。ロールには、CodeCatalystWorkflowDevelopmentRole-*spaceName*一意の識別子が追加された名前が付けられます。ロールとロールポリシーの詳細については、「」を参照してくださいCodeCatalystWorkflowDevelopmentRole-***spaceName*** サービスロールについて。ロールを作成する手順については、「」を参照してくださいアカウントとスペースのCodeCatalystWorkflowDevelopmentRole-***spaceName***ロールの作成。ロールがアカウントに追加され、 のプロジェクト作成ページに表示されます CodeCatalyst。

の Amazon CodeCatalyst Spaces ページにアクセスできない AWS Management Console

問題： の Amazon CodeCatalyst ページにアクセスして、アカウントを CodeCatalyst スペースAWS Management Consoleに追加したり、 のアカウントにロールを追加しようとするとAWS、アクセス許可エラーが発生します。

解決方法:

Space administrator ロールがある場合は、スペースをプロジェクトAWS アカウントに追加する権限を追加できます。また、管理者権限AWS アカウントを持っているか、AWS管理者と連携できる も必要です。まず、管理するのと同じアカウントAWS Management Consoleで にサインインしていることを確認する必要があります。にサインインしたらAWS Management Console、 コンソールを開いて再試行できます。

https://us-west-2.console.aws.amazon.com/codecatalyst/home?region=us-west-2#/ の で Amazon AWS Management Console CodeCatalyst ページを開きます。

請求アカウントとは異なるアカウントが必要

問題： CodeCatalyst ログインを設定するときに、スペースを設定し、承認された を関連付けるためのいくつかのステップを完了しましたAWS アカウント。次に、別のアカウントに請求を承認します。

解決方法： Space 管理者ロールがある場合は、スペースに対して請求アカウントを承認できます。また、管理者権限AWS アカウントを持っているか、AWS管理者と連携できる も必要です。

詳細については、「Amazon CodeCatalyst 管理者ガイド」の「請求の管理」を参照してください。

開発環境の問題のトラブルシューティング

開発環境に関連する問題をトラブルシューティングするには、以下のセクションを参照してください。開発環境の詳細については、「」を参照してください。で開発環境を使用してコードを記述および変更する CodeCatalyst

トピック

- クォータの問題により、開発環境の作成が成功しませんでした。
- Dev Environment からリポジトリ内の特定のブランチに変更をプッシュできない

- 開発環境が再開されなかった
- 開発環境が切断されました
- VPC に接続した開発環境に障害が発生しました
- 自分のプロジェクトがどのディレクトリにあるのか分からない
- SSH 経由で開発環境に接続できません
- ローカル SSH 設定がないため、SSH 経由で開発環境に接続できません。
- プロファイルに問題があるため、SSH 経由で開発環境に接続できません。AWS Configcodecatalyst
- IDEsトラブルシューティング
- devfile に関する問題のトラブルシューティング

クォータの問題により、開発環境の作成が成功しませんでした。

問題:で開発環境を作成したいのですが CodeCatalyst、エラーが表示されます。コンソールの Dev Environments ページに、スペースのストレージ制限に達したというメッセージが表示されます。

解決方法:プロジェクトまたはスペースでの役割に応じて、独自の Dev Environment を 1 つ以上削除できます。また、スペース管理者権限を持っている場合は、他のユーザーが作成した未使用の Dev Environment を削除できます。また、課金レベルをよりストレージの多いレベルに変更することもできます。

- ストレージ制限を確認するには、Amazon CodeCatalyst スペースの [請求] タブを表示して、使用クォータが最大許容値に達しているかどうかを確認してください。クォータが上限に達した場合は、スペース管理者ロールを持つ担当者に連絡して、不要な Dev Environment を削除するか、請求範囲の変更を検討してください。
- 作成した開発環境のうち、不要になったものを削除する方法については、を参照してください。開発環境の削除

問題が解決せず、IDE でエラーが発生する場合は、Dev Environment CodeCatalyst を作成できるロールがあるかどうかを確認してください。スペース管理者ロール、プロジェクト管理者ロール、コントリビューターロールにはすべて Dev Environment を作成する権限があります。詳細については、「ユーザーロールによるアクセス許可の付与」を参照してください。

Dev Environment からリポジトリ内の特定のブランチに変更をプッシュできない

問題:Dev Environment のコード変更をソースリポジトリのブランチにコミットしてプッシュしたいのですが、エラーが表示されます。

考えられる解決方法:プロジェクトまたはスペースでの役割によっては、プロジェクト内のソースリポジトリにコードをプッシュする権限がない場合があります。スペース管理者ロール、プロジェクト管理者ロール、および寄稿者ロールはすべて、プロジェクト内のリポジトリにコードをプッシュする権限を持っています。

Contributor ロールはあるが特定のブランチにコードをプッシュできない場合は、そのロールを持つユーザーがその特定のブランチにコードをプッシュできないようにするブランチルールが特定のブランチに設定されている可能性があります。変更を別のブランチにプッシュしてみるか、ブランチを作成してコードをそのブランチにプッシュしてみてください。詳細については、「ブランチルールを使用してブランチに対して許可されるアクションを管理する」を参照してください。

開発環境が再開されなかった

問題:開発環境を停止しても再開しませんでした。

考えられる解決方法:問題を解決するには、Amazon CodeCatalyst スペースの [請求] タブを表示して、使用割り当てが上限に達しているかどうかを確認してください。クォータが上限に達した場合は、スペース管理者に連絡して請求範囲を引き上げてください。

開発環境が切断されました

問題:使用中に開発環境が切断されました。

解決方法:問題を解決するには、インターネット接続を確認してください。インターネットに接続していない場合は、Dev Environment に接続して作業を再開してください。

VPC に接続した開発環境に障害が発生しました

問題:VPC 接続を開発環境に関連付けたところ、エラーが発生します。

考えられる解決方法:Docker はブリッジネットワークと呼ばれるリンク層デバイスを使用して、同じブリッジネットワークに接続されているコンテナ間の通信を可能にします。デフォルトのブリッジは、コンテナのネットワークに通常 `172.17.0.0/16` サブネットを使用します。環境のインスタン

スの VPC サブネットが、Docker で既に使用しているのと同じアドレス範囲を使用している場合、IP アドレスの競合が発生する可能性があります。Amazon VPC が同じ IPv4 CIDR Docker アドレスブロックを使用しているために発生する IP アドレスの競合を解決するには、とは異なる CIDR ブロックを設定します。`172.17.0.0/16`

> **Note**
>
> 既存の VPC またはサブネットの IP アドレス範囲は変更できません。

自分のプロジェクトがどのディレクトリにあるのか分からない

問題:自分のプロジェクトがどのディレクトリにあるのか分からない。

解決方法:プロジェクトを見つけるには、ディレクトリをに変更してください`/projects`。これはプロジェクトを見つけることができるディレクトリです。

SSH 経由で開発環境に接続できません

SSH 経由の Dev Environment への接続をトラブルシューティングするには、`ssh -vvv`オプション付きのコマンドを実行すると、問題の解決方法に関する詳細情報を表示できます。

```
ssh -vvv codecatalyst-dev-env=<space-name>=<project-name>=<dev-environment-id>
```

ローカル SSH 設定がないため、SSH 経由で開発環境に接続できません。

ローカル SSH 設定 (`~/.ssh/config`) が見つからなかったり、`Host codecatalyst-dev-env*`セクションの内容が古くなっていたりすると、SSH 経由で開発環境に接続できなくなります。この問題を解決するには、`Host codecatalyst-dev-env*`セクションを削除し、SSH Access モーダルから最初のコマンドをもう一度実行してください。詳細については、「SSH を使用した開発環境への接続」を参照してください。

プロファイルに問題があるため、SSH 経由で開発環境に接続できません。AWS Config`codecatalyst`

`codecatalyst`プロファイルの AWS Config (`~/.aws/config`) が、で説明されているものと一致していることを確認してくださいAWS CLIとを使用するためのセットアップ CodeCatalyst。一致しない場合は、`codecatalyst`のプロファイルを削除し、SSH アクセスモーダルから最初のコマンドを

もう一度実行してください。詳細については、「SSH を使用した開発環境への接続」を参照してください。

IDEsトラブルシューティング

以下のセクションを参照して、 IDEs に関連する問題のトラブルシューティングを行いますCodeCatalyst。IDEs「」を参照してくださいIDE で開発環境を作成する。

トピック

- でランタイムイメージのバージョンが一致しません AWS Cloud9
- /projects/projects の にあるファイルにアクセスできない AWS Cloud9
- カスタム devfile AWS Cloud9を使用して で開発環境を起動できない
- に問題がある AWS Cloud9
- では JetBrains、 を使用して開発環境に接続できません。 CodeCatalyst
- IDE AWS Toolkitに をインストールできません
- IDE で開発環境を起動できない

でランタイムイメージのバージョンが一致しません AWS Cloud9

AWS Cloud9 は、フロントエンドアセットとバックエンドランタイムイメージの異なるバージョンを使用しています。異なるバージョンを使用すると、Git 拡張機能 と が正しく動作AWS Toolkitしなくなる可能性があります。この問題を解決するには、開発環境ダッシュボードに移動し、開発環境を停止してから、もう一度起動します。APIs、 `UpdateDevEnvironment` API を使用してランタイムを更新します。詳細については、「Amazon CodeCatalyst API リファレンスUpdateDevEnvironment」の「」を参照してください。

`/projects/projects` の にあるファイルにアクセスできない AWS Cloud9

AWS Cloud9 エディタはディレクトリ 内のファイルにアクセスできません`/projects/projects`。この問題を解決するには、AWS Cloud9ターミナルを使用してファイルにアクセスするか、別のディレクトリに移動します。

カスタム devfile AWS Cloud9を使用して で開発環境を起動できない

devfile イメージは と互換性がない可能性がありますAWS Cloud9。この問題を解決するには、リポジトリと対応する開発環境から devfile を確認し、新しい開発環境を作成して続行します。

に問題がある AWS Cloud9

その他の問題については、AWS Cloud9「 ユーザーガイド」の「トラブルシューティング」セクションを確認してください。

では JetBrains、 を使用して開発環境に接続できません。 CodeCatalyst

この問題を解決するには、 の最新バージョンのみ JetBrains がインストールされていることを確認します。複数のバージョンがある場合は、古いバージョンをアンインストールし、IDE とブラウザを閉じてプロトコルハンドラーを再度登録します。次に、 を開き JetBrains 、プロトコルハンドラーを再度登録します。

IDE AWS Toolkitに をインストールできません

VS Code のこの問題を解決するには、 AWS Toolkit for Visual Studio Codeから を手動でインストールしますGitHub。

のこの問題を修正するには JetBrains、 AWS Toolkit for JetBrainsから を手動でインストールしますGitHub。

IDE で開発環境を起動できない

VS Code のこの問題を修正するには、VS Code の最新バージョンと AWS Toolkit for Visual Studio Codeがインストールされていることを確認します。最新バージョンがない場合は、開発環境を更新して起動します。詳細については、「Amazon CodeCatalyst for VS Code」を参照してください。

のこの問題を修正するには JetBrains、 JetBrains と の最新バージョンAWS Toolkit for JetBrainsがインストールされていることを確認します。最新バージョンがない場合は、開発環境を更新して起動します。詳細については、「Amazon CodeCatalyst for JetBrains」を参照してください。

devfile に関する問題のトラブルシューティング

の devfiles に関連する問題のトラブルシューティングについては、以下のセクションを参照してください。 CodeCatalystdevfiles の詳細については、を参照してください。開発環境用の devfile の設定

トピック

- カスタム devfile にカスタムイメージを実装しているのに、私の開発環境ではデフォルトのユニバーサル devfile が使用されています。
- 私のプロジェクトは、デフォルトのユニバーサル devfile を使用して開発環境でビルドされていません

カスタム devfile にカスタムイメージを実装しているのに、私の開発環境ではデフォルトのユニバーサル devfile が使用されています。

カスタム devfile CodeCatalyst を使用する開発環境の起動中にエラーが発生した場合、開発環境はデフォルトでデフォルトのユニバーサル devfile に設定されます。問題を解決するには、以下のログで正確なエラーを確認できます。`/aws/mde/logs/devfile.log`post`Start`実行が成功したかどうかは、ログで確認することもできます`/aws/mde/logs/devfileCommand.log`。

私のプロジェクトは、デフォルトのユニバーサル devfile を使用して開発環境でビルドされていません

問題を解決するには、カスタム devfile を使用していないか確認してください。カスタム devfile を使用していない場合は、`devfile.yaml`プロジェクトのソースリポジトリにあるファイルを表示して、エラーを見つけて修正してください。

開発環境のリポジトリ開発ファイルを移動したいです。

デフォルトの devfile `/projects/devfile.yaml` をソースコードリポジトリに移動できます。devfile の場所を更新するには、以下のコマンドを使用します。`/aws/mde/mde start --location` *`repository-name`*`/devfile.yaml`

devfile の起動に問題があります。

devfile の起動に問題がある場合はリカバリモードに入り、引き続き環境に接続して devfile を修正できます。リカバリーモードでは、実行中に `/aws/mde/mde status` devfile の場所は記録されません。

```
{
    "status": "STABLE"
}
```

下のログでエラーを確認し`/aws/mde/logs`、devfile を修正して、`/aws/mde/mde start`もう一度実行してみてください。

devfile のステータスを確認する方法がわかりません。

devfile のステータスは実行することで確認できます。`/aws/mde/mde status`このコマンドを実行すると、次のいずれかが表示される場合があります。

- `{"status": "STABLE", "location": "devfile.yaml" }`

  これは devfile が正しいことを示しています。
- `{"status": "STABLE" }`

  これは devfile が起動できず、リカバリモードに入ったことを示しています。

`/aws/mde/logs/devfile.log`正確なエラーは下のログで確認できます。

`postStart`実行が成功したかどうかは、ログで確認することもできます:`/aws/mde/logs/devfileCommand.log`.

詳細については、「開発環境用のユニバーサル devfile イメージの指定」を参照してください。

私の開発ファイルは、最新のイメージで提供されているツールと互換性がありません

お使いの Dev Environment では、`latest`特定のプロジェクトに必要なツールがツールに含まれていない場合、`devfile`devfile `postStart`またはツールが失敗する可能性があります。問題を解決するには、以下を実行してください。

1. 開発ファイルに移動します。
2. devfile で、代わりに詳細なイメージバージョンに更新してください。`latest`以下のようになるかもしれません。

   ```
   components:
     - container:
         image: public.ecr.aws/amazonlinux/universal-image:1.0
   ```
3. 更新した devfile を使用して新しい開発環境を作成します。

問題と問題のトラブルシューティング

以下の情報は、の問題に関する一般的な問題のトラブルシューティングに役立ちます。CodeCatalyst

トピック

問題の担当者が選べません。

問題:課題を作成するときに、担当者のリストが空です。

考えられる解決策:担当者のリストは、 CodeCatalyst プロジェクトのメンバーとしてリストされているユーザーに直接リンクされています。ユーザープロファイルへのアクセスが正しく機能していることを確認するには、プロファイルアイコンを選択し、次に [ユーザープロファイル] を選択します。ユーザープロファイル情報が入力されない場合は、ヘルスレポートにインシデントがないか確認してください。入力された場合は、サービスチケットを提出してください。

Amazon CodeCatalyst と AWS SDK 間の問題のトラブルシューティング AWS CLI

以下の情報は、AWS CLIや SDK を使用する際によく発生する問題のトラブルシューティングに役立ちます。 CodeCatalyst AWS

トピック

aws codecatalystコマンドラインまたはターミナルで入力すると、「選択が無効です」というエラーが表示されます。

問題:AWS CLIwith を使おうとすると CodeCatalyst、1 aws codecatalyst つ以上のコマンドが有効と認識されません。

解決策:この問題の最も一般的な原因は、AWS CLI使用しているバージョンのが最新のサービスやコマンドに対応していないことです。AWS CLIのインストールを更新して、もう一度試してください。詳細については、AWS CLIとを使用するためのセットアップ CodeCatalystを参照してください。

aws codecatalystコマンドを実行すると認証情報エラーが表示されます。

問題:AWS CLIwith を使用しようとすると CodeCatalyst、`You can configure credentials by running "aws configure".Unable to locate authorization token`またはというメッセージが表示されます。

解決策:AWS CLI CodeCatalyst コマンドと連動するようにプロファイルを設定する必要があります。詳細については、「AWS CLIとを使用するためのセットアップ CodeCatalyst」を参照してください。

CodeCatalyst ヘルスレポートを使用した現在のサービスステータスの理解

Amazon CodeCatalyst ヘルスレポートは、広範な影響を与える のサービスのリソースパフォーマンスと可用性に関する up-to-the-minute 通知の集計リスト CodeCatalyst をユーザーに提供するパブリックダッシュボードです。どのリソースに問題があり、 のアプリケーションに影響を与える可能性があるかを確認できます CodeCatalyst。これにより、システム全体の停止やその他のリソースのダウンタイムを追跡できます。インシデントが発生すると、ヘルスレポートアイコンに青いインジケータが表示されます。さらに、 は、プロジェクト内のスペース管理者ロールを持つすべてのユーザーにアラートと E メール通知 CodeCatalyst を自動的に送信し、インシデントの詳細と履歴をほぼリアルタイムで提供します。

ダッシュボードには、すべてのアクティブなイベントのリストと、過去 30 日間に発生した最大 100 件の以前のインシデントの記録が表示されます。インシデントのリストは、インシデントが更新された日付に基づいて整理できます。また、インシデントのリストを最大 1 分間更新できます。

CodeCatalyst ヘルスレポートを使用するためのワークフローを次に示します。

Mateo Jackson は、スペース管理者権限を持つバディングスペースの開発者です。プルリクエストの作成中に、エラーメッセージが表示され続けます。彼は E メールをチェックし、自分のスペースに影響するシステムの問題に関する詳細な履歴 CodeCatalyst を提供した から自動生成されたシステムインシデント E メールを受け取ったことを発見しました。更新の表示を選択すると、 CodeCatalyst ヘルスレポートが表示され、システム報告のすべてのインシデントを表示できます。リストからインシデントを選択して、詳細を確認します。分割画面が開き、インシデントの最終更新、履歴、影響を受ける機能、開始時刻、および現在のステータスのタイムスタンプが表示されます。また、問題が進行中であることも確認できますが、サービスチームはその作業を開始しています。インシデントの履歴またはステータスが更新されるたびに、彼は E メールを受け取ります。E メールにアクセスできない場合は、上部パネルのベルアイコンを選択して CodeCatalyst ヘルスレポートにアクセスできます。

CodeCatalyst ヘルスレポートの概念

以下の概念を学ぶと、 CodeCatalyst ヘルスレポートと、アプリケーション、サービス、リソースのヘルスを追跡する方法を理解するのに役立ちます。

インシデント

インシデントは、 内のアプリケーションとリソースに影響を与えているシステムイベントです CodeCatalyst。インシデントを選択すると、開始時刻やサービスチームが解決に取り組んでいるかどうかなど、イベントの詳細な履歴を表示できます。

ステータス

ステータスは、インシデントのリアルタイムステータスです。「進行中」または「解決済み」と表示されます。

影響を受ける機能

影響を受ける機能は、インシデントの影響を受けるリソースまたはアプリケーションです。1 つのインシデントが、プルリクエスト 、問題 、ワークフロー 、テスト 、デプロイ 、ソース など、システム内の複数の領域に影響を与える可能性があります。

更新日

の更新は、インシデントの最終更新のタイムスタンプを提供します。

AWS SupportAmazon 用 CodeCatalyst

スペースを作成するときは、AWS アカウントを接続してスペースの請求先アカウントとして指定する必要があります。請求先アカウントとして指定した場所から、Amazon AWS Support CodeCatalyst のプランにアクセスすることもできます。AWS アカウントサポートが必要な場合は、AWS アカウント指定されたアカウントからサポートケースを作成できます。

CodeCatalyst スペース内のユーザーは AWS Support for Amazon CodeCatalyst ページを使用してサポートケースを管理します。 CodeCatalyst ビジネスSupport やエンタープライズSupport AWS Support などのプランにアップグレードして、 CodeCatalyst でテクニカルサポートケースを作成および管理できます CodeCatalyst。Support ケースについては、電話、Web、またはチャットを通じてサポートを受けることができます。

AWS Supportfor Amazon を通じてサポートできるのは、 CodeCatalyst サービスとリソースに固有のケースのみです CodeCatalyst。 CodeCatalyst リソースには、 CodeCatalyst AWS内部またはユーザーによってデプロイされたリソースが含まれますが CodeCatalyst、他のサービスやサードパーティのサービスにデプロイされたリソースは含まれません。AWS他のサービスのサポートが必要な場合は、を通じてサポートを開始する必要がありますAWS Management Console。

サポートプランを変更するには、「サポートプランの変更」を参照してください。

> **Note**
>
> デベロッパーSupport プランは、実稼働環境向けには設計されていません。スペース請求アカウントに開発者Support プランがある場合、このプランは内のすべてのスペース管理者およびスペースメンバーにはカスケードされません。AWS Support CodeCatalyst

Amazon AWS Support ンの請求 CodeCatalyst

でスペースを作成すると CodeCatalyst、そのスペースのユーザーは AWS Support for Amazon からサポートケースを作成および管理できます CodeCatalyst。次の 2 種類のカスタマーケースを作成できます。

- アカウントと請求に関するサポートケースは、 CodeCatalyst スペース内のすべてのユーザーが利用できます。請求やアカウントに関する質問は、での権限に基づいてサポートを受けることができます CodeCatalyst。

- テクニカルサポートケースを利用すると、テクニカルサポートエンジニアに連絡して、サービス関連の技術的な問題やサードパーティアプリケーションの拡張についてサポートを受けることができます。Basic Support プランをご利用の場合は、技術サポートケースを作成できません。

  AWS アカウントスペースの請求先として指定されたアカウントには、AWS Support CodeCatalyst 技術的なケースに使用するスペースのビジネスSupport プランまたはエンタープライズSupport プランが必要です。

  > **Note**
  >
  > ビジネスSupport プランやエンタープライズSupport CodeCatalyst プランに加入していないアカウントから Amazon AWS Support AWS Support 用のスペースを使用している場合でも、アカウントや請求のケースには Amazon CodeCatalyst 用を使用できます。

テクニカルサポートでは、 CodeCatalyst すべてのケースをコンソールから開く必要があります。CodeCatalyst AWS Supportからテクニカルサポートケースを作成することはできませんAWS Management Console。

> **Note**
>
> Amazon では、サービス制限の引き上げリクエストは受け付けていません CodeCatalyst。AWS Supportこれらのリクエストは、のスペース請求アカウントの root ユーザーのみが送信できますAWS Support Center Console。

AWS Supportfor Amazon CodeCatalyst にはと同じサポート契約がありますがAWS Support、以下の点が考慮されます。

- for のサポートケースには、「重要度の選択」で詳しく説明されているように、AWS Supportに記載されている重要度リスト CodeCatalyst、応答時間、および SLA AWS Support が適用されます。
- スペース管理者やスペースメンバーは、Slack の AWS Support API、AWS SDK、AWS Supportアプリを使用してのケースを作成することはできません。 CodeCatalyst CodeCatalyst CodeCatalyst サポートケースはからのみ送信できます。

> **Note**
>
> CodeCatalyst AWS Trusted AdvisorAWSまたはインシデント検出および対応と完全には統合されていません。 CodeCatalyst どのように統合されているかを検証して、貴社のビジネス慣行が現在の統合と一致していることを確認してください。

サポートをリクエストしたい分野のユーザーである必要があります。

> **Note**
>
> スペースに複数のビルダーがいる場合は、ビジネスSupport またはエンタープライズSupport プランを購入することをお勧めします。これらのプランは、最大 5,000 人のビルダーを対象にスペースのテクニカルサポートを提供します。

AWS アカウントスペースの請求アカウントとして指定されたユーザーは、`AWSRoleForCodeCatalystSupport`AmazonCodeCatalystSupportAccessロールと管理ポリシーを使用します。これにより、 CodeCatalyst スペース内のユーザーは AWS Support for Amazon CodeCatalyst ページにアクセスできるようになります。このロールとポリシーの詳細については、を参照してくださいAmazonCodeCatalystSupportAccess。請求に関するその他の考慮事項については、『Amazon CodeCatalyst 管理者ガイド』の「請求の管理」を参照してください。

CodeCatalystビルダーがサポートケースを作成する場合のフローは次のとおりです。

Mateo Jackson は、にあるプロジェクトの開発者です。 CodeCatalystAmazon AWS アカウント CodeCatalyst の請求を管理するにサインアップし、ビジネスSupport プランにアップグレードすると、この分野のすべてのビルダーがテクニカルサポートケースを作成できます。AWS SupportMateo は、プロジェクトで失敗したワークフローのテクニカルサポートケースを提出します。Mateo は AWS Support for Amazon CodeCatalyst ページを使用してフォームに記入し、リクエストにワークフロー ID とその他の詳細情報を入力してケースを作成します。ケースはケース ID で作成され、AWS アカウント請求先アカウントとして指定され、スペースのサポートプランに関連付けられているアカウント ID が含まれます。

AWS Supportfor ではすべてのビルダーがサポートケースを作成できますが CodeCatalyst、ケースが作成されるたびに課金されることはありません。Space AWS Support 請求アカウントで購入したプレミアムプランに基づいて、ケースと連絡先を事実上無制限に開くことができます。

> **Note**
>
> スペース請求アカウントは、AWS アカウント CodeCatalyst ユーザーとリソースに対して課金されるアカウントです。他のサービスにデプロイした場合はAWS アカウント、AWS Supportを通じて連絡し、AWS Management Console他のサービスにデプロイされたリソースについてサポートを受けてください。
> AWS アカウントどのデプロイ先をワークフローから特定できます。

Amazon AWS Support 用のスペースをセットアップする CodeCatalyst

AWS Supportfor CodeCatalyst Amazonは、AWS Support CodeCatalystとのAPI統合の一環としてサポートケースを管理しています。

`AWSRoleForCodeCatalystSupport`このロールは、スペース内のサポートケースに使用されるサービスロールです。このロールは、スペースに指定されている請求先アカウントに追加する必要があります。詳細やロールの作成については、を参照してくださいアカウントとスペースのAWSRoleForCodeCatalystSupportロールの作成。

> **Note**
>
> 2023 年 4 月 20 日より前に作成されたスペースでは、 CodeCatalyst サポートが自分のスペースで機能するようにロールを作成する必要があります。2023 年 4 月 20 日以降にスペースを作成する場合は、スペース作成時にの「請求詳細」ページ CodeCatalyst、または内のサポートバナーリンクをクリックしてロールを作成できます。 CodeCatalyst

スペースのサポートを設定するには

1. CodeCatalyst スペースを作成すると、請求先アカウントを接続するように指示されます。スペースに指定されている請求先アカウントには、が請求されます。AWSスペースの作成に関する詳細は、を参照してください最初のスペースと開発ロールの作成 (招待なしで開始)。
2. CodeCatalyst スペースを作成すると、`AWSRoleForCodeCatalystSupport` CodeCatalyst ユーザーがサポートにアクセスできるようにするサービスロールを作成するオプションが表示されます。このロールは管理ポリシーを使用しますAmazonCodeCatalystSupportAccess。ロールは、AWS アカウントスペースの請求アカウントとして指定されたアカウントに追加

する必要があります。このロールの作成に関する詳細については、「アカウントとスペースのAWSRoleForCodeCatalystSupportロールの作成」をご参照ください。

3. スペースの指定請求アカウントについては、スペース管理者はのビジネスSupport またはエンタープライズSupport プランを購入することをお勧めしますAWS アカウント。このスペースのメンバーは全員 AWS Support for Amazon からサポートケースを管理できるようになりCodeCatalyst、AWS Supportサポートチャネルは統合が完了したプランに合わせて調整されます。
4. でサポートケースを作成および管理するには CodeCatalyst、を参照してください。CodeCatalyst でのサポートケースの作成 CodeCatalyst

CodeCatalyst でのサポートへのアクセス AWS Management Console

スペースのサポートが有効な請求アカウントが切断されると、AWS Support以前のスペース請求アカウントと関連するサポートプランに関連付けられているケースは AWS Support for Amazon CodeCatalyst に表示されなくなります。その請求アカウントのルートユーザーは、から古いケースを閲覧して解決できます。また、他のユーザーが古いケースを閲覧して解決できるように IAM 権限を設定できます。AWS Management Console AWS SupportAWS のサービスその他のすべてのサポートプランの特典を引き続き活用したり、 CodeCatalyst 以前に解決されていないサポートケースを処理したりすることはできます。AWS Management Console

詳細については、『ユーザーガイド』の「ケースの更新、解決、再開」を参照してください。AWS Support

CodeCatalyst に関する一般的なハウツー情報のSupport ケースもで開くことができますがAWS Management Console、このチャネルではテクニカルサポートを受けることができません。CodeCatalyst詳細については、『AWS Supportユーザーガイド』の「サポートケースの作成とケース管理」を参照してください。

以下は、ユーザーが以下のサポートケースを解決する際に考えられるフローです。 CodeCatalyst AWS Management Console

すべてのビルダーが AWS Support for Amazon でサポートケースを作成できますが CodeCatalyst、サポートリクエストはスペースの請求アカウントとして指定されたアカウントから請求されます。Mateo Jackson はプロジェクトの開発者で、 CodeCatalyst そのプロジェクトで失敗したワークフローのテクニカルサポートケースをオープンしました。ただし、Amazon CodeCatalyst にサイン

アップし、ビジネスSupport プランを購入したスペースの請求先アカウントは、スペースから切断されています。AWS SupportMateo が最新のコミュニケーションを確認し、未解決のケースを解決する唯一の方法は、AWS Supportのセンターからケース ID CodeCatalyst を管理することです。AWS Management Consoleそのために、Mateo はサポートケースに添付されていた以前のスペース請求アカウントのルートユーザーから IAM 権限を与えられ、コンソールでケースを解決します。AWS Support

> **Important**
>
> スペースの指定請求アカウントを変更した場合でも、月末までは「のみ」AWS Support を通じてプランにアクセスできます。AWS Management ConsoleAWS Supportで以前に作成したサポートケースに引き続きアクセスするには、 CodeCatalyst更新された請求先アカウントで再購入する必要があります。AWS Supportfor Amazon を通じてサポートケースにアクセスすることへの影響を避けるため、 CodeCatalystすべてのサポートケースが解決されるまで待ってからスペース請求アカウントを変更することをおすすめします。

CodeCatalyst でのサポートケースの作成 CodeCatalyst

AWS Supportfor Amazon CodeCatalyst のページでサポートケースを作成できます。

AWSBuilder ID は、認証に使用したエイリアスと、その権限に基づくリソースについてのみサポートを受けることができます。アカウントと請求オプションは、すべてのスペース管理者とスペースメンバーが使用できます。ただし、ユーザーはアクセスできるリソースについてのみサポートを受けることができ、アカウントの請求管理に関するサポートは受けられません。 CodeCatalyst

CodeCatalyst スペースのフォームページを使用して、 CodeCatalyst AWS Supportアカウントと請求に関するケースまたはリソースに関するテクニカルサポートケースを作成できます。

> **Note**
>
> AWS Supportfor Amazon を通じてサポートできるのは、 CodeCatalyst サービスとリソースに固有のケースのみです CodeCatalyst。 CodeCatalyst リソースには、 CodeCatalyst AWS 内部またはユーザーによってデプロイされたリソースが含まれますが CodeCatalyst、他のサービスやサードパーティのサービスにデプロイされたリソースは含まれません。AWS他のサービスのサポートが必要な場合は、を通じてサポートを開始する必要がありますAWS Management Console。

サポートケースを作成するには、 CodeCatalyst

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. CodeCatalyst 自分のスペースに移動します。

> **Tip**
>
> 複数のスペースに属している場合は、上部のナビゲーションバーでスペースを選択します。

3. ページ上部の [?] を選択します。 アイコンをクリックし、[Support] を選択します。
4. [Create case (ケースを作成)] を選択します。
5. 以下のオプションのいずれかを選択します。

   - アカウントと請求
   - 技術的

> **Note**
>
> AWS Supportfor Amazon では CodeCatalyst、ビジネスSupport またはエンタープライズSupport プランをスペース請求アカウントに追加すると、 CodeCatalyst すべてのスペース管理者とスペースメンバーがテクニカルケースサポートを利用できるようになります。トラブルシューティング情報については、「自分のスペースのテクニカルサポートケースを作成できない」を参照してください。
>
> AWS Supportプランは複数のスペースにまたがることはありません。複数のスペースに所属している場合、スペース管理者がすべてのスペースでテクニカルサポートを受けるには、AWS Supportスペース管理者が各スペースのプレミアムプランを購入する必要があります。

6. [サービス]、[カテゴリ]、および [緊急度] を選択します。重要度の選択について詳しくは、「重要度の選択」を参照してください。

   - 一般的な質問、または機能要望
   - 問題発生中、または開発中の急ぎの問い合わせ
   - 本番環境の重要な機能に障害発生中
   - 本番環境のシステム停止中
   - 本番環境のビジネスクリティカルなシステム停止中

7. [Next step: Additional information] (次のステップ:追加情報) を選択します。
8. 追加情報ページの件名に、問題に関するタイトルを入力します。
9. 説明では、プロンプトに従って、次のようにケースを説明します。
   - ワークフロー ID、ログ CodeCatalyst、スクリーンショットなど、固有のトラブルシューティング情報
   - 受信したエラーメッセージ
   - 使用したトラブルシューティング手順

   > **Note**
   >
   > 認証情報、クレジットカード、署名付き URL、個人を特定できる情報など、機密情報をケース通信で共有しないでください。

10. (オプション) 添付ファイルを選択して、エラーログやスクリーンショットなど、関連するファイルをケースに追加します。最大 3 つまでのファイルをアタッチできます。各ファイルは、最大 5MB まで可能です。
11. [スペース名] には、スペースの名前が表示されます。
12. 「ビルダー名」には、AWSビルダー ID に関連付けられているフルネームが自動的に入力されます。
13. (オプション) [プロジェクト名] でプロジェクトを選択します (該当する場合)。

    > **Note**
    >
    > 権限のあるプロジェクトのみが表示されます。別のプロジェクトへのアクセス権が必要な場合は、サポートケースを作成する前に、プロジェクト管理者にアクセス権の付与を依頼してください。

14. [次のステップ:お問い合わせ] を選択します。
15. [優先連絡言語] で、デフォルトを選択します。現時点では英語のみご利用いただけます。
16. 連絡方法として、ウェブ、電話、またはチャットオプションを選択します。
17. ケースの詳細を確認し、[送信] を選択します。ケース ID 番号と概要が表示されます。

サポートケースはスペースレベルで作成され、サポートケースで定義されているスペースとプロジェクト (選択した場合) にアクセスできるすべてのメンバーが閲覧できます。現時点では、個々のユーザーからのサポートケースを省略する方法はありません。

でのサポートケースの解決 CodeCatalyst

未解決のサポートケースは、AWS Support for CodeCatalyst Amazonページから解決できます。

サポートケースを解決したいスペースには、スペース管理者またはスペースメンバーロールが必要です。スペース管理者権限がない場合や、ケースの作成時にプロジェクトが選択されていた場合は、ケースを確認して解決するには、そのプロジェクトのメンバーシップも必要です。

で未解決のサポートケースを解決するには CodeCatalyst

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. CodeCatalyst 自分のスペースに移動します。

> **Tip**
>
> 複数のスペースに属している場合は、上部のナビゲーションバーでスペースを選択します。

3. ページ上部の [?] を選択します。 アイコンをタップし、「Amazon」を選択しますAWS Support CodeCatalyst。
4. 管理したいサポートケースのリンクを選択します。[Resolve case] (ケースを解決) を選択します。

のサポートケースを再開する CodeCatalyst

AWS Supportfor CodeCatalyst Amazonページから、解決済みのサポートケースを再オープンできます。

> **Note**
>
> 再度開くことができるのは、問題が解決されてから 14 日以内のサポートケースです。ただし、14 日以上非アクティブなケースを再度開くことはできません。ケースを再開できない場合は、新しいケースを開き、以前のケース ID を参考として含めてください。

現在の問題とは異なる情報を持つ既存のケースを再度開いた場合、サポート担当者が新しいケースを作成するよう依頼することがあります。

サポートケースを再度開くには CodeCatalyst

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. CodeCatalyst 自分のスペースに移動します。

> **Tip**
>
> 複数のスペースに属している場合は、上部のナビゲーションバーでスペースを選択します。

3. ページ上部の [?] を選択します。 アイコンをクリックし、[AWS Support] を選択します CodeCatalyst。
4. 管理したいサポートケースのリンクを選択します。[Reopen] (再オープン) を選択します。確認画面で [OK] を選択し、次に [Submit] を選択します。
5. 同じ問題に関する最新情報を説明に入力します。クレデンシャル、クレジットカード、署名付き URL、個人を特定できる情報など、機密情報をケース通信で共有しないでください。

のクォータ CodeCatalyst

次の表に、Amazon のクォータと制限を示します CodeCatalyst。の特定の側面に関する追加情報は、以下のトピック CodeCatalyst で確認できます。

- のソースリポジトリのクォータ CodeCatalyst
- での ID、アクセス許可、アクセスのクォータ CodeCatalyst
- ワークフローのクォータ
- の開発環境のクォータ CodeCatalyst
- プロジェクトのクォータ
- のブループリントのクォータ CodeCatalyst
- スペースのクォータ
- の問題のクォータ CodeCatalyst

アカウントのスペースの最大数	5
ユーザーが 1 か月に作成できるスペースの最大数	5
スペース AWS アカウント の最小数	1
スペースのアカウント接続の最大数	5,000
スペースの請求アカウント AWS アカウント としての の最大数	1
スペースの VPC 接続の最大数	100
スペース内のプロジェクトの最大数	100
ユーザーが属できるプロジェクトの最大数	1,000
スペースの説明	スペースの説明はオプションです。指定した場合、0～200 文字の長さである必要があります。文字、数字、スペース、ピリオド、アンダースコア、カンマ、ダッシュ、および次の特

	殊文字の任意の組み合わせを含めることができます。 ? & $ % + = / \ ; : \n \t \r
スペース名	スペース名は 全体で一意である必要があります CodeCatalyst。削除されたスペースの名前は再利用できません。 スペース名の長さは 3～63 文字にする必要があります。また、英数字で始まる必要があります。スペース名には、文字、数字、ピリオド、アンダースコア、ダッシュを任意に組み合わせて使用できます。次の文字を含めることはできません。 ! ? @ # $ % ^ & * ( ) + = { } [ ] | 八 > < ~ ` ' " ; :

Amazon のドキュメント履歴 CodeCatalyst

次の表は、 のドキュメント全体のドキュメント履歴と更新を示しています CodeCatalyst。

変更	説明	日付
新しいコンテンツ: 個人用接続を使用する手順を追加	個人用接続を作成および削除する手順を追加しました。個人用接続を使用すると、Amazon のプロジェクトとブループリントの GitHub リソースを管理できます CodeCatalyst。	2024 年 6 月 6 日
更新されたコンテンツ: ブループリントでサードパーティーのリポジトリを使用する	、、プロジェクトにブループリントを適用してリソースを追加するまたはカスタムブループリントの作成時に 設計図を使用したプロジェクトの作成 GitHub または Bitbucket リポジトリを作成できるようにドキュメントを更新しました。	2024 年 6 月 5 日
新しいコンテンツ: Bitbucket リポジトリ拡張機能	で Bitbucket リポジトリ拡張機能を使用するための新しいコンテンツを追加しました CodeCatalyst。	2024 年 6 月 5 日
新しいコンテンツ: アクションタイプ	SonarCloud スキャンアクションについて言及するようにサードパーティーアクショントピックを更新しました。	2024 年 5 月 8 日
更新された内容: AWS CDK「デプロイ」アクション YAML 定義	`CdkRootRootPath` 例を修正しました。	2024 年 5 月 28 日

更新された内容	読みやすさと検出を向上させるために、トピックタイトルを更新し、コンテンツを再編成しました。これらの変更に関するフィードバックを提供する場合は、「フィードバックを提供する」リンクを使用します。	2024 年 5 月 17 日
新しいコンテンツ: ファイルの変更履歴の表示	ソースリポジトリ内のファイルへの変更履歴を表示するための新機能を反映するようにドキュメントを更新しました。	2024 年 5 月 1 日
更新されたコンテンツ: チュートリアル: 生成 AI 機能の使用	Amazon Q Developer との統合を反映するようにチュートリアルを更新しました。	2024 年 4 月 29 日
更新されたコンテンツ: チュートリアル: 生成 AI 機能の使用	チュートリアルを更新して、Amazon Q が問題の複雑さを分析し、タスクを提案して作成し、問題のあるタスクを処理できるようにしました。	2024 年 4 月 22 日
新しいコンテンツ: ワークフロー実行のゲート設定	ワークフローの承認に関連する ワークフロー実行のゲート設定ワークフロー実行の承認を要求する、、およびその他のいくつかのトピックを追加しました。	2024 年 4 月 22 日

新しいコンテンツ: チュートリアル: 構成可能な PDK ブループリントを使用したフルスタックアプリケーションの作成	Amazon プロジェクトで AWS Project Development Kit (AWS PDK) ブループリントを使用するための新しいチュートリアルを追加しました CodeCatalyst 。	2024 年 4 月 9 日
新しいコンテンツ: タスクを使用して問題を小さな目標に分割する	問題でのタスクの起動をサポートするコンテンツを追加しました。タスクを問題に追加して、その問題の作業をさらに分類、整理、追跡できます。	2024 年 4 月 4 日
更新された内容: アーティファクトを使用したワークフロー内のアクション間のデータの共有	アーティファクトを使用したワークフロー内のアクション間のデータの共有 トピックを更新して、 アーティファクトを出力と入力として指定せずに共有できますか？と の 2 つの新しいサブトピックを追加しました。 ワークフロー間でアーティファクトを共有できますか？	2024 年 4 月 2 日
更新された内容: での GitHub アクションの制限 CodeCatalyst	古いランタイム環境の Docker イメージで実行されている GitHubアクションを示すでの GitHub アクションの制限 CodeCatalystようにトピックを更新しました。	2024 年 4 月 2 日
新しいコンテンツ: AWS CDK「デプロイ」アクション YAML 定義	に新しいCloudAssemblyRootPath プロパティを追加しましたAWS CDK「デプロイ」アクション YAML 定義。	2024 年 4 月 1 日

更新された内容: ランタイム環境の Docker イメージの指定	ランタイム環境の Docker イメージの指定 トピックを更新して、2024 年 3 月のランタイム環境イメージに関する情報を追加しました。	2024 年 3 月 26 日
更新されたコンテンツ: ロールの使用	ロールのアクセス許可情報を 1 つのテーブルに統合しました。テーブルは新しい各ロールで使用できるアクセス許可の表示トピックにあります。	2024 年 3 月 18 日
新しいコンテンツ: ユーザーのすべてのスペースとプロジェクトを表示する	でサインインしたユーザーの各 CodeCatalyst スペースまたはプロジェクトを表示するユーザーホームページのリストの表示に関する情報を追加しました CodeCatalyst。ユーザーのすべてのスペースとプロジェクトを表示する を参照してください。	2024 年 3 月 18 日
新しいコンテンツ: 例: プルとブランチを含むトリガー	プルリクエストトリガーの例を追加しました。トリガーを使用したワークフローの自動実行の開始 トピック全体で小さな修正を行いました。	2024 年 3 月 11 日
更新されたコンテンツ: ロールの使用	環境を作成、削除、表示するためのアクセス許可を含めるようにロールのドキュメントを更新しました。	2024 年 3 月 4 日
更新されたコンテンツ: チュートリアル: 生成 AI 機能の使用	問題を作成して Amazon Q に割り当てる際の変更を反映するようにチュートリアルを更新しました。	2024 年 3 月 4 日

新しいコンテンツ: コンポーネントを発行する	カスタムブループリント開発者として問題コンポーネントを操作する方法に関する新しいコンテンツを追加しました。	2024 年 2 月 27 日
更新された内容: アクションタイプ	CodeCatalyst Labs アクションのリストを含めるようにCodeCatalyst ラボアクショントピックを更新しました。	2024 年 2 月 21 日
更新されたコンテンツ: プルリクエストの使用	新しい機能を承認ルールとともに反映し、プルリクエストをマージするための要件を上書きするようにドキュメントを更新しました。	2024 年 2 月 15 日
更新されたコンテンツ: プルリクエストのマージ	必要なレビューワーからの承認をまだ受け取っていない、または承認ルールを満たしていないプルリクエストをマージするためのマージ要件の上書きに関する情報を含むプルリクエストのドキュメントを追加しました。	2024 年 2 月 15 日
新しいコンテンツ: 承認ルールの管理	承認ルールの作成と管理に関する情報を含むプルリクエストのドキュメントを追加しました。	2024 年 2 月 15 日
更新されたコンテンツ: ロールの使用	承認ルールとプルリクエストを操作するためのアクセス許可を含めるようにロールのドキュメントを更新しました。	2024 年 2 月 14 日

更新された内容: 「ワークフロー定義に *n* エラーがある」エラーを修正するにはどうすればよいですか？	トラブルシューティングのヒントを追加するために、「ワークフロー定義に *n* エラーがある」エラーを修正するにはどうすればよいですか？セクションを更新しました。	2024 年 2 月 9 日
新しいコンテンツ: ワークフローステータスの表示	ワークフローの状態を説明するセクションを追加しました。	2024 年 2 月 9 日
新しいコンテンツ: ワークフローステータスの表示	ワークフローの状態を説明するセクションを追加しました。	2024 年 2 月 9 日
コンテンツの更新: ワークフローのクォータ	ワークフローあたりのアクションの最大数と、スペースごとのクォータに関連付けられた環境の最大数でワークフローのクォータトピックを更新しました。 AWS アカウント	2024 年 2 月 7 日
更新された内容: 環境を作成する	環境ごとに最大 1 つのアカウント接続を使用できることを示す 環境を作成するセクションを更新しました。	2024 年 1 月 31 日
新しいコンテンツ: カスタムブループリント GitHub リポジトリ	公開される GitHub リポジトリの新しいコンテンツを追加しました。	2024 年 1 月 10 日
更新されたコンテンツ: で npm を設定する CodeCatalyst	で npm を使用するための一般的な設定手順を更新し CodeCatalyst、 `always-auth=true` オプションに関する説明を追加しました。	2024 年 1 月 5 日

更新されたコンテンツ: プルリクエストの使用	の生成 AI 機能で新機能を反映するようにドキュメントを更新しました CodeCatalyst。	2023 年 11 月 28 日
更新されたコンテンツ: 問題の作成	の生成 AI 機能で新機能を反映するようにドキュメントを更新しました CodeCatalyst。	2023 年 11 月 28 日
新しいコンテンツ: チュートリアル: 生成 AI 機能の使用	Amazon の生成 AI 機能を使用するためのチュートリアルを追加しました CodeCatalyst。	2023 年 11 月 28 日
新しいコンテンツ: カスタムブループリントとライフサイクル管理	Amazon でカスタムブループリントとライフサイクル管理機能を使用するための新しいコンテンツを追加しました CodeCatalyst。	2023 年 11 月 27 日
更新された内容: チュートリアル:モダン 3 層 Web アプリケーションブループリントによるプロジェクトの作成	修正とトラブルシューティング情報を含むチュートリアルを更新しました。	2023年11月22日
更新された内容: トリガーを使用したワークフローの自動実行の開始	プルリクエストトリガーに関連するいくつかの例と説明を修正しました。分岐時のトリガーに関する考慮事項 セクションを追加しました。	2023年11月22日

新しいコンテンツ: SSO でサインイン	Single Sign-On (SSO) でのサインインに関する情報と、ID フェデレーションをサポートする CodeCatalyst スペースのセットアップと管理に関する情報へのリンクを追加しました。「のセットアップと へのサインイン CodeCatalyst」および「SSO でサインイン」を参照してください。	2023 年 11 月 17 日
更新されたコンテンツ: ロールの使用	チーム、VPC 接続、シングルサインオン、マシンリソースを操作するためのアクセス許可を含めるようにロールのドキュメントを更新しました。	2023 年 11 月 16 日
更新されたコンテンツ: プルリクエストの使用	プルリクエストの変更の表示方法の変更を反映するようにドキュメントを更新しました。	2023 年 11 月 16 日
更新されたコンテンツ: のクォータ CodeCatalyst	スペースクォータの VPC 接続の最大数でのクォータ CodeCatalystトピックを更新しました。	2023 年 11 月 16 日
新しいコンテンツ: スペースと CodeCatalyst プロジェクトのチームの管理	スペースでのチームの使用に関する情報を追加しました。「チームを使用したスペースアクセスの許可」および「チームを使用したプロジェクトアクセスの許可」を参照してください。	2023 年 11 月 16 日

新しいコンテンツ: スペース内の設計図とワークフローのマシンリソースの管理	スペースでのマシンリソースの使用に関する情報を追加しました。マシンリソースのスペースアクセスを許可する を参照してください。	2023 年 11 月 16 日
新しいコンテンツ: CodeCatalyst プロジェクトの設計図とワークフローのマシンリソースの管理	CodeCatalyst プロジェクトでのマシンリソースの使用に関する情報を追加しました。マシンリソースへのプロジェクトアクセスを許可する を参照してください。	2023 年 11 月 16 日
新しいコンテンツ: VPC 接続を環境に関連付ける	ワークフローで使用できる環境と VPC 接続を関連付けるためのドキュメントを追加しました。	2023 年 11 月 16 日
新しいコンテンツ: VPC 接続を開発環境に関連付ける	VPC 接続で開発環境を使用するためのドキュメントを追加しました。	2023 年 11 月 16 日
新しいコンテンツ	Amazon CodeCatalyst 管理者ガイド の初回発行。	2023 年 11 月 16 日
新しいコンテンツ: AWS CDK「デプロイ」アクション YAML 定義	AWS CDK「デプロイ」アクション YAML 定義 と に新しい`CdkCliVersion` プロパティを追加しましたAWS CDK「ブートストラップ」アクション YAML 定義。	2023 年 11 月 14 日
更新されたコンテンツ: ロールの使用	ロールのドキュメントを更新し、ブランチルールを操作するためのアクセス許可を追加しました。	2023 年 11 月 13 日

更新されたコンテンツ: ソースリポジトリ、ワークフロー、開発環境に関する問題のトラブルシューティング	トラブルシューティングトピックを更新し、ブランチルールの使用に関する情報を追加しました。	2023 年 11 月 13 日
更新された内容: ビルドおよびテストアクションの YAML 定義	`Environment` プロパティのドキュメントを更新しました。ビルドアクションとテストアクションのオプションフィールドになりました。	2023 年 11 月 13 日
新しいコンテンツ: ブランチルールの管理	ソースリポジトリ内のブランチのルールの表示、およびブランチルールの作成と管理に関する情報を含むブランチのドキュメントを追加しました。	2023 年 11 月 13 日
更新されたコンテンツ: プルリクエストの使用	プルリクエストに関する情報の表示方法の変更を反映するようにドキュメントを更新しました。	2023 年 11 月 10 日
更新された内容: ワークフロー実行間のファイルのキャッシュ	ファイルキャッシュの制限を含めるようにドキュメントを更新しました。	2023 年 11 月 10 日
更新された内容: チュートリアル: Amazon EKS にアプリケーションをデプロイする	EKS アプリケーションデプロイの設計図について言及するようにドキュメントを更新しました。	2023 年 11 月 9 日
新しいコンテンツ: のパッケージ CodeCatalyst	でパッケージを使用するためのドキュメントを追加しました CodeCatalyst。	2023 年 11 月 1 日

新規および更新されたコンテンツ: ロールの使用	の 4 つの新しいロールのドキュメントを更新しました CodeCatalyst: パワーユーザー、制限付きアクセス、レビュー担当者、読み取り専用。	2023 年 11 月 1 日
更新された内容: 他のアクションで使用できるように GitHub 出力パラメータをエクスポートする	`set-output` コマンドの代わりに`GITHUB_OUTPUT` 環境ファイルを使用するように例を更新しました。環境ファイルを使用することは、出力パラメータを設定するための GitHubの推奨方法です。	2023 年 10 月 24 日
新しいコンテンツ: トリガーを使用したワークフローの自動実行の開始	スケジュールトリガーに関するドキュメントを追加しました。	2023 年 10 月 16 日
更新された内容: 「Kubernetes クラスターにデプロイ」アクション YAML 定義	`CodeCatalystWorkflowDevelopmentRole-`*`spaceName`* ロールの使用に関する情報を 「Kubernetes クラスターにデプロイ」アクション YAML 定義および チュートリアル: Amazon EKS にアプリケーションをデプロイするトピックに追加しました。	2023 年 9 月 22 日

更新されたコンテンツ: ロールの新しいCodeCatalystWorkflowDevelopmentRole-*spaceName* ロール名とポリシー	デベロッパーロール名の変更に関するステップとロールの説明を に更新しましたCodeCatalystWorkflowDevelopmentRole-*spaceName* 。開発者ロールは AdministratorAccess AWS マネージドポリシーを使用するようになりました。「CodeCatalystWorkflowDevelopmentRole-*spaceName* サービスロールについて」および「最初のスペースと開発ロールの作成 (招待なしで開始)」を参照してください。	2023 年 9 月 20 日
更新された内容: ワークフローでの変数の設定と使用	ユーザー定義変数 と事前定義された変数 の 2 つの新しい概念が導入されました。これらの概念により、 ワークフローでの変数の設定と使用セクションを読みやすく、理解しやすくなるはずです。	2023 年 9 月 19 日
更新されたコンテンツ: コミットの使用	表示される情報の変更を反映し、複数の親とのコミットの表示に関する詳細を提供するようにドキュメントを更新しました。	2023 年 9 月 7 日
新しいコンテンツ: トリガーを使用したワークフローの自動実行の開始	トリガーを使用したワークフローの自動実行の開始 トピックに次の例を追加しました。例: シンプルな「メインへのプッシュ」トリガー	2023 年 9 月 6 日

更新されたコンテンツ: プルリクエストの使用	プルリクエストの作成時にソースブランチと送信先ブランチの表示順序の変更を反映するようにドキュメントを更新しました。	2023 年 8 月 30 日
新しいコンテンツ: デフォルトのブランチを表示および変更する	ソースリポジトリのデフォルトブランチの表示と変更に関する情報を含むブランチのドキュメントを追加しました。	2023 年 8 月 30 日
更新された内容: 「Kubernetes クラスターにデプロイ」アクション YAML 定義	Helm と Kustomize に関するメモを のManifestsプロパティの説明に追加しました「Kubernetes クラスターにデプロイ」アクション YAML 定義。	2023 年 8 月 15 日
新しいコンテンツ: 問題添付ファイルの管理	問題に対する添付ファイルの操作と管理に関するドキュメントを追加しました。	2023 年 8 月 15 日
更新された内容: トリガーを使用したワークフローの自動実行の開始	ワークフロートリガーに関連するドキュメントを改善し、拡張しました。	2023 年 8 月 11 日
新しいコンテンツ: ロールのアクセス許可のトラブルシューティング	Amazon へのアクセスを必要とするワークフローを実行するためのロールのアクセス許可の更新に関する情報を追加しました CodeGuru。OnPullRequest最新の3層ウェブアプリケーションブループリントワークフローがAmazonの権限エラーで失敗する CodeGuru を参照してください。	2023 年 8 月 11 日

新しいコンテンツ: 「ワークフローは非アクティブです」というメッセージを修正するにはどうすればよいですか？	次のトラブルシューティングトピックを追加しました。「ワークフローは非アクティブです」というメッセージを修正するにはどうすればよいですか？	2023 年 8 月 11 日
新しいコンテンツ: ワークフローを使用した Amazon Elastic Kubernetes Service へのアプリケーションのデプロイ	Kubernetes クラスターへのデプロイアクションのドキュメントを追加しました。詳細については、「ワークフローを使用した Amazon Elastic Kubernetes Service へのアプリケーションのデプロイ」および「チュートリアル: Amazon EKS にアプリケーションをデプロイする」を参照してください。	2023 年 7 月 27 日
CodeCatalyst スペースの管理イベントの記録方法の更新	を使用して、 CodeCatalystスペース内の特定のアクションの管理イベントがどのように記録されるかに関する情報を追加しました AWS CloudTrail。 list-event-logs コマンドを使用してスペース内のすべてのイベントを表示する方法に関する情報を追加しました。ログ記録を使用したイベントと API コールのモニタリング を参照してください。	2023 年 7 月 20 日

更新された内容: トリガーを使用したワークフローの自動実行の開始	プルリクエストトリガーが GitHub ソースリポジトリでサポートされるようになったことを示すドキュメントを更新しました。以前は、プルリクエストトリガーは CodeCatalyst ソースリポジトリでのみサポートされていました。	2023 年 7 月 14 日
更新された内容: ワークフローのクォータ	アクションがクォータを実行できる最大時間でワークフローのクォータトピックを更新しました。	2023 年 6 月 27 日
更新された内容: ワークフロー YAML 定義	Compute コードブロックのフォーマットエラーを修正しました。	2023 年 6 月 27 日
更新されたコンテンツ: データ保護	データレプリケーションに関する追加情報を含めるようにドキュメントを更新しました。	2023 年 6 月 26 日
新しいコンテンツ: アクションのメジャー、マイナー、またはパッチバージョンの指定	アクションのメジャー、マイナー、またはパッチバージョンの指定 トピックを追加しました。	2023 年 6 月 21 日
更新された内容: 環境を使用して AWS アカウント および VPCsにデプロイする CodeCatalyst	環境をサポートするアクションはどれですか？ セクションを明確化しました。	2023 年 6 月 14 日
内容の更新: 問題ドキュメントの再編成	ドキュメントセットとユーザーフロー全体に合わせて、問題のドキュメントのほとんどを再編成しました。	2023 年 5 月 31 日

更新されたコンテンツ: Issues View Switcher	更新された問題ビュースイッチャーに合わせて、さまざまなユーザーフローを更新しました。	2023 年 5 月 31 日
更新されたコンテンツ: 通知の管理	通知に関するドキュメントを更新し、個人用の Slack 通知の設定に関する情報を追加しました。	2023 年 5 月 30 日
更新されたコンテンツ: 通知の管理	通知に関するドキュメントを更新し、個人用の Slack 通知の設定に関する情報を追加しました。	2023 年 5 月 30 日
新しいコンテンツ : CodeCatalyst 信頼モデル	接続された で がサービスロールを引き受け CodeCatalyst ることができる信頼モデルに関する情報を含む新しいトピックを追加しました AWS アカウント。に定義されたサービスプリンシパルに関する新しいセクションを追加しました CodeCatalyst。CodeCatalyst 信頼モデルを理解する を参照してください。	2023 年 5 月 20 日
更新された内容: ワークフローをソースリポジトリに接続する	の手順を簡素化しましたソースリポジトリ内のファイルを参照する。	2023 年 5 月 10 日
更新された内容: ワークフローのクォータ	出力変数値クォータの最大長でワークフローのクォータトピックを更新しました。	2023 年 5 月 10 日

新しいコンテンツ: アーティファクトを使用したワークフロー内のアクション間のデータの共有	と の 2 つの例を追加例: 単一のアーティファクトでファイルを参照するしました例: WorkflowSource が存在する場合にアーティファクト内のファイルを参照する。	2023 年 5 月 10 日
更新されたコンテンツ: プルリクエストの使用	プルリクエストのドキュメントを更新し、プルリクエストイベントの E メール設定に関する情報を追加しました。	2023 年 4 月 21 日
更新されたコンテンツ: 通知の管理	プルリクエストイベントの E メール設定の設定に関する情報を含む通知のドキュメントを更新しました。	2023 年 4 月 21 日
マネージドポリシーの更新	AWS マネージドポリシー: AmazonCodeCatalystFullAccess、AWS マネージドポリシー: AmazonCodeCatalystReadOnlyAccess、AWS マネージドポリシー: AmazonCodeCatalystSupportAccess管理ポリシーを追加しました。AWS マネージドポリシーに関する CodeCatalyst の更新 を参照してください。	2023 年 4 月 20 日
新しいコンテンツ: デプロイターゲットの削除	デプロイターゲットの削除 トピックを追加しました。	2023 年 4 月 20 日
新しいコンテンツ: アクションタイプ	CodeCatalyst アクション トピックを追加しました。	2023 年 4 月 20 日

スペース内のスペース管理者ロールを持つユーザーを管理するための更新	スペース内のスペース管理者ロールを持つユーザーのロールの削除または変更に関する情報を追加しました。Space 管理者ロールを持つユーザーのロールの削除または変更 を参照してください。	2023 年 4 月 19 日
開発環境を管理するための更新	開発環境をスペース管理者として管理する方法についての情報を追加しました。スペースの開発環境の管理 を参照してください。	2023 年 4 月 19 日
更新されたコンテンツ: 問題の検出と表示	問題の検出と表示 トピックとサブトピックを再編成しました。	2023 年 4 月 19 日
更新された内容: ワークフローのコンピューティング環境とランタイム環境の Docker イメージの設定	Amazon Linux 2 での Arm64 アーキテクチャのサポートが追加されました。	2023 年 4 月 19 日
新しいコンテンツ: グループ内での問題の移動	ボードビューとすべての問題ビューのグループ内で問題を移動するためのドキュメントを追加しました。	2023 年 4 月 19 日
更新された内容: ワークフローのクォータ	クォータが欠落しているワークフローのクォータトピックを更新し、単一アクションの出力変数クォータの最大合計サイズを 120 KB (2 KB から) に更新しました。	2023 年 4 月 18 日
新しいコンテンツ: アクションのソースコードの表示	アクションのソースコードの表示 トピックを追加しました。	2023 年 4 月 18 日

新しいコンテンツ: レポートのテストケースの再試行	レポートのテストケースの再試行 トピックを追加しました。	2023 年 4 月 11 日
新しいコンテンツ: ワークフロー実行の停止	ワークフロー実行の停止 トピックを追加しました。	2023 年 4 月 10 日
新しいコンテンツ: AWS と Amazon 間のアカウント接続のリソースにタグ付けするためのセクションを追加 CodeCatalyst	アカウント接続リソースのタグ付けと、接続リソースの IAM ポリシーの管理に関する情報を追加しました。「タグを使用してアカウント接続リソースへのアクセスを制御する」および「CodeCatalyst アクセス許可リファレンス」を参照してください。	2023 年 4 月 6 日
新しいコンテンツ: アクションタイプ	アクションタイプ トピックを追加しました。	2023 年 4 月 6 日
更新された内容: AWS CloudFormation 「スタックのデプロイ」アクション YAML 定義	`parameter-overrides` プロパティの説明を更新しました。JSON ファイルをサポートするようになりました。	2023 年 4 月 5 日
新しいコンテンツ: リンクされた GitHub リポジトリ CodeCatalyst を使用して でプロジェクトを作成する	GitHub リポジトリにリンクするプロジェクトを作成する手順「プロジェクトの作成」リンクされたサードパーティーリポジトリを使用したプロジェクトの作成が記載された新しいセクションを に追加しました。	2023 年 4 月 5 日

更新されたコンテンツ: 通知の使用	通知に関するドキュメントを更新し、プロジェクトイベントに関する E メールの設定に関する情報を追加しました。	2023 年 3 月 31 日
新しいコンテンツ	Amazon CodeCatalyst Action Development Kit ガイド の初回公開。	2023 年 3 月 31 日
コンテンツの更新: Amazon の Spaces セクションを再構築しました CodeCatalyst	ランディングページを削除し、トピックを統合することで、スペースセクションを更新しました。	2023 年 3 月 29 日
更新された内容: チュートリアル: Amazon ECS にアプリケーションをデプロイする	AWS IAM Identity Center の代わりに でユーザーを作成する方法をステップ 1: AWS ユーザーと を設定する AWS CloudShell記述するように変更しました AWS Identity and Access Management。IAM ユーザーの作成は推奨されなくなりました。	2023 年 3 月 23 日
更新されたコンテンツ: ロールの使用	Space 管理者、プロジェクト管理者、および寄稿者ロールのドキュメントを更新し、問題をプルリクエストにリンクするためのアクセス許可を追加しました。	2023 年 3 月 13 日
更新されたコンテンツ: プルリクエストの使用	プルリクエストのドキュメントを更新し、問題をプルリクエストにリンクする情報を追加しました。	2023 年 3 月 13 日

更新されたコンテンツ: 問題の処理	問題に関するドキュメントを更新し、問題とプルリクエストのリンクに関する情報を追加しました。	2023 年 3 月 13 日
新しいコンテンツ: プロジェクト内のすべての実行のステータスと詳細の表示	新しい集約ワークフロー実行ページを説明するセクションを追加しました。	2023 年 3 月 8 日
新しいコンテンツ: 「ワークフロー定義に *n* エラーがある」エラーを修正するにはどうすればよいですか？	「ワークフロー定義にエラーがある」エラーのトラブルシューティング方法に関するセクションを追加しました。	2023 年 3 月 7 日
更新された内容: ワークフローの作成	新しい UI を反映するように手順を更新しました。	2023 年 3 月 3 日
新しいコンテンツ: テストアクション universal-test-runner への統合	テストアクション universal-test-runner への統合 トピックを追加しました。	2023 年 3 月 3 日
更新された内容: でワークフローを使用して構築、テスト、デプロイする CodeCatalyst	ワークフローの概要ページで新しいソースリポジトリ、ブランチ、ワークフロー名フィルターを反映するように、さまざまなセクションを更新しました。	2023 年 3 月 2 日
新しいコンテンツ: コミットによるデプロイステータスの追跡	コミットごとのコード品質とデプロイステータスの表示に関するセクションを追加しました。	2023 年 2 月 27 日
新しいコンテンツ: ソースによって生成される変数 (BranchName「」およびCommidId「」)	新しいBranchName 事前定義された変数を追加しました。	2023 年 2 月 16 日

更新されたコンテンツ: Amazon でのスペースメンバーの管理 CodeCatalyst	でユーザーに割り当てられたロールに基づいて、2 つの新しいテーブルのメンバーロールの変更、メンバーの招待、メンバーの削除に関する情報を更新しました CodeCatalyst。	2023 年 2 月 15 日
更新されたコンテンツ: Amazon CodeCatalyst コンソールで PAT 管理の手順を追加	コンソールで PATs を表示、作成、削除する手順を追加しました。	2023 年 2 月 15 日
更新された内容: ランタイム環境の Docker イメージの指定	デフォルトのイメージツールのバージョン表にツールを追加しました。	2023 年 1 月 10 日
更新された内容: アーティファクトを使用したワークフロー内のアクション間のデータの共有	アーティファクトパスを修正しました。	2023 年 1 月 3 日
更新された内容: GitHub 「アクション」アクション YAML 定義	Steps セクションのコードスニペットを修正しました。	2023 年 1 月 3 日
更新された内容: ワークフローをソースリポジトリに接続する	ソースパスを修正しました。	2023 年 1 月 3 日
更新されたコンテンツ: プルリクエストの更新	プルリクエストに必要なレビューワーまたはオプションのレビューワーの更新に関する情報を含むようにドキュメントを更新しました。	2022 年 12 月 23 日

新しいコンテンツ: ワークフロー実行間のファイルのキャッシュ	ワークフローにファイルキャッシュのページを追加しました。	2022 年 12 月 20 日
更新されたコンテンツ: プルリクエストの使用	プルリクエストのドキュメントを更新し、通知に関する情報を追加しました。	2022 年 12 月 16 日
新しいコンテンツ: AWS CDK「デプロイ」アクション YAML 定義	新しいCdkRootPath プロパティを追加しました。	2022 年 12 月 16 日
新しいコンテンツ: アクション間でのコンピューティングの共有	アクション間でのコンピューティングの共有 トピックを追加しました。	2022 年 12 月 14 日
更新された内容: アーティファクトを使用したワークフロー内のアクション間のデータの共有	入力アーティファクトを指定する方法を示す例を修正しました。	2022 年 12 月 13 日
新しいコンテンツ: GitHub「アクション」アクション YAML 定義	GitHub Actions アクション専用のリファレンスページを追加しました。	2022 年 12 月 13 日
更新されたコンテンツ: のプロジェクトのクォータ CodeCatalyst	1 つのスペースに最大 100 のプロジェクトを含むドキュメントを更新しました。	2022 年 12 月 2 日
新しいコンテンツ	Amazon CodeCatalyst ユーザーガイドの初回発行。	2022 年 12 月 1 日

AWS 用語集

最新の AWS 用語については、「 AWS の用語集 リファレンス」のAWS 「 用語集」を参照してください。

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。