aws

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 コンセプト
- AWS Builder ID ユーザーをサポートするスペースの作成
- 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 コンセプト

Amazon でのコラボレーションとアプリケーション開発をスピードアップするのに役立つ重要な概念を理解してください CodeCatalyst。これらの概念には、ソース管理、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ル化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 ビルダー ID のスペース CodeCatalyst
- の 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 コネクション
- 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
- パーソナルアクセストークン (PAT)
- ロール

AWS ビルダー ID のスペース CodeCatalyst

CodeCatalyst スペース管理者はメンバーページから個別の招待メールを送信してユーザーを招待します。招待されたユーザ、またはサインアップして自分の AWS Builder ID CodeCatalyst を作成したユーザ。 AWS プロファイルはBuilder IDで管理され、のユーザー設定にユーザー名とプロファイル情報として表示されます CodeCatalyst。

の 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、アイデンティティストアで管理され、IAM Identity Center を通じてスペースに招待されたユーザー。 CodeCatalyst スペース管理者はメンバーページを同期して最新の更新情報を確認します。ユーザーは、会社の IAM Identity Center インスタンスに設定されている 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 へのマッピングを通じて 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、 CodeCatalyst 開発チームとタスクをサポートする共同作業です。プロジェクトを作成したら、ユーザーやリソースを追加、更新、削除したり、プロジェクトダッシュボードをカスタマイズしたり、チームの作業の進捗状況を監視したりできます。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のプロジェクト 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同時に、 CodeCatalyst アプリケーションサポートファイルや依存関係を生成して拡張する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、プロジェクトリポジトリと生成されるリソースをプレビューします。プロジェクトは、ブループリントで指定されている基本設定から生成されます。プロジェクトブループリントと定期的に統合します。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、リソースが再生されます。プロジェクトでは 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することでプロジェクトを統合します。これらのファイルには`package.json`、アプリケーションの種類と言語に応じて`Makefile`、`eslint`、、、などが含まれる場合があります。

プロジェクトブループリントでは、CDK 構成、 AWS CloudFormation テンプレート、 AWS テンプレートなどのリソースをサポートするファイルを生成できます。 AWS Serverless Application Model

プロジェクトブループリントの詳細については、を参照してください。プロジェクトブループリントリファレンス

アカウント接続

アカウント接続により、 CodeCatalyst スペースが自分に関連付けられます。 AWS アカウントアカウント接続が設定されると、 AWS アカウント はスペースで使用できるようになります。その後、IAM CodeCatalyst ロールをに追加して、スペース内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 CodeCatalyst これらのロールはワークフローアクションにも使用できます。

アカウント接続の詳細については、を参照してくださいAWS アカウント スペースの管理。

VPC コネクション

VPC 接続は、VPC CodeCatalyst にアクセスするためのワークフローに必要なすべての設定を含むリソースです。スペース管理者は、スペースメンバーに代わって Amazon CodeCatalyst コンソールで独自の 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たり、ネットワークルールに準拠した開発環境を作成したり、関連する VPC 内のリソースにアクセスしたりできます。

VPC 接続の詳細については、『CodeCatalyst 管理者ガイド』の「Amazon 仮想プライベートクラウドの管理」を参照してください。

AWS ビルダー ID

AWS ビルダー ID は、他の参加アプリケーションへのサインアップやサインインに使用できる個人の ID です。 CodeCatalyst とは同じではありません AWS アカウント。 AWS Builder ID は、ユーザーエイリアスやメールアドレスなどのメタデータを管理します。 AWS Builder ID は、内のすべてのスペースのユーザーをサポートする固有の ID CodeCatalyst です。 AWS Builder ID プロファイルへのアクセス方法については、を参照してくださいプロフィールを更新する。 AWS ビルダー ID の詳細については、の「AWS ビルダー 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セットアップ 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、の任意のページのログインイニシャル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個人アクセストークン (PAT) はプロフィールページから作成できますが、PAT の表示や削除はを使用してしか行えません。 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。ユーザー名は変更できません。 CodeCatalyst 別のユーザーのプロフィー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。

AWS Builder ID にアクセスするには、 CodeCatalyst プロフィールを表示して AWS Builder ID に移動することを選択します。 AWS Builder ID のプロフィールページにリダイレクトされます。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 AWS Builder ID によって管理され、Builder ID ページを使用してその情報を編集できます。 AWS この情報はサインアップ時に入力しました。ログ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、 AWS ビルダー ID ページを使用します。 AWS Builder ID プロファイルの表示について詳しくは、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セットアップ 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、 CodeCatalyst ソースリポジトリは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は複数のソースリポジトリを設定できます。でプロジェクトのソースリポジトリを作成できます。また CodeCatalyst、インストール済みのエクステンションでそのサービスがサポートされている場合は、別のサービスがホストす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。たとえば、 GitHub GitHub Repositories エクステンションをインストールした後に、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、「でのソースリポジトリの操作 CodeCatalyst」および「クイックスタート: GitHub でのリポジトリの使用 CodeCatalyst」を参照してください。

ソースリポジトリには、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、CodeCatalystプロジェクトの設定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、プロジェクト設定情報が保存されたソースリポジトリが作成されます。空のプロジェクトを作成した場合、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セットへの変更です。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、変更を加えたユーザーの ID、変更の日時、コミットのタイトル、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 でのコミットの処理 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。ユーザが変更をコミットしてプッシュするたびに、変更をコミットしたユーザ、コミットの日時、 CodeCatalyst コミットの一部として加えられた変更などの情報が保存されます。特定のコミットを識別しやすくするために、コミットに Git タグを追加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 でのコミットの処理 CodeCatalyst

開発環境

Dev Environment はクラウドベースの開発環境で CodeCatalyst 、これ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すばやく操作できます。Dev Environment 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にある 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 が自動的に適用されます。デフォルト 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、および 16 GiB の永続ストレージを搭載するように構成されています。

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と継続的デリバリー (CI/CD) システムの一部としてコードをビルド、テスト、デプロイする方法を記述した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、つまりアクションを定義します。ワークフローでは、ワークフローを開始させるイベント、つまりトリガーも定義されます。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エディターまたは YAML エディター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、ブループリントを使用してプロジェクトを作成してください。各ブループリントには、レビュー、実行、実験が可能な、機能するワークフローが導入されています。

ワークフローの詳細については、「のワークフローによるビルド、テスト、デプロイ 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論理的な作業単位を定義します。通常、ワークフローには、設定方法に応じて順次またはparallel 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アクションの使用。

問題

課題とは、プロジェクトに関連する作業を記録する記録です。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課題を作成できます。アジャイル開発を使用している場合は、課題にエピックやユーザーストーリーを説明することもできます。

課題の詳細については、を参照してください。の問題点 CodeCatalyst

パーソナルアクセストークン (PAT)

パーソナルアクセストークン (PAT) はパスワードに似ています。ユーザー ID と関連付けられ、内のすべてのスペースとプロジェクトで使用できます。 CodeCatalystPAT を使用して、統合開発環境 (IDE) や Git CodeCatalyst ベースのソースリポジトリを含むリソースにアクセスします。PAT CodeCatalyst はユーザーを代表するもので、ユーザー設定で管理できます。1 人のユーザが複数の PAT を持つことができます。個人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、必ずローカルコンピューターに安全に保管してください。デフォルトでは、PAT は 1 年後に有効期限が切れます。

PAT の詳細については、を参照してください。Amazon での個人アクセストークンの管理 CodeCatalyst

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権と、そのユーザーが実行できるアクションを定義します。ユーザーをプロジェクトに招待するときに、そのユーザーのロールを選択します。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者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。たとえば、あるプロジェクトのプロジェクト管理者権限を持つユーザーは、そのプロジェクトを完全に制御でき、そのプロジェクト内のユーザーの役割を変更できます。どのロールが利用可能で、各ロールにどのような権限があるかについては、を参照してくださいAmazon のロールを使った作業 CodeCatalyst。

ロールの詳細については、「Amazon のロールを使った作業 CodeCatalyst」をご参照ください。

セットアップ CodeCatalyst

設定できるスペースには、 AWS Builder ID ユーザーをサポートするスペースと、IAM Identity Center で SSO ユーザーとグループを管理する ID フェデレーションをサポートするスペースの作成という 2 種類のスペースがあります。 CodeCatalyst AWS Builder ID スペースのユーザーは自分の AWS Builder ID を使用してサインインし、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する会社の SSO CodeCatalyst ポータルを使用してサインインします。 CodeCatalyst

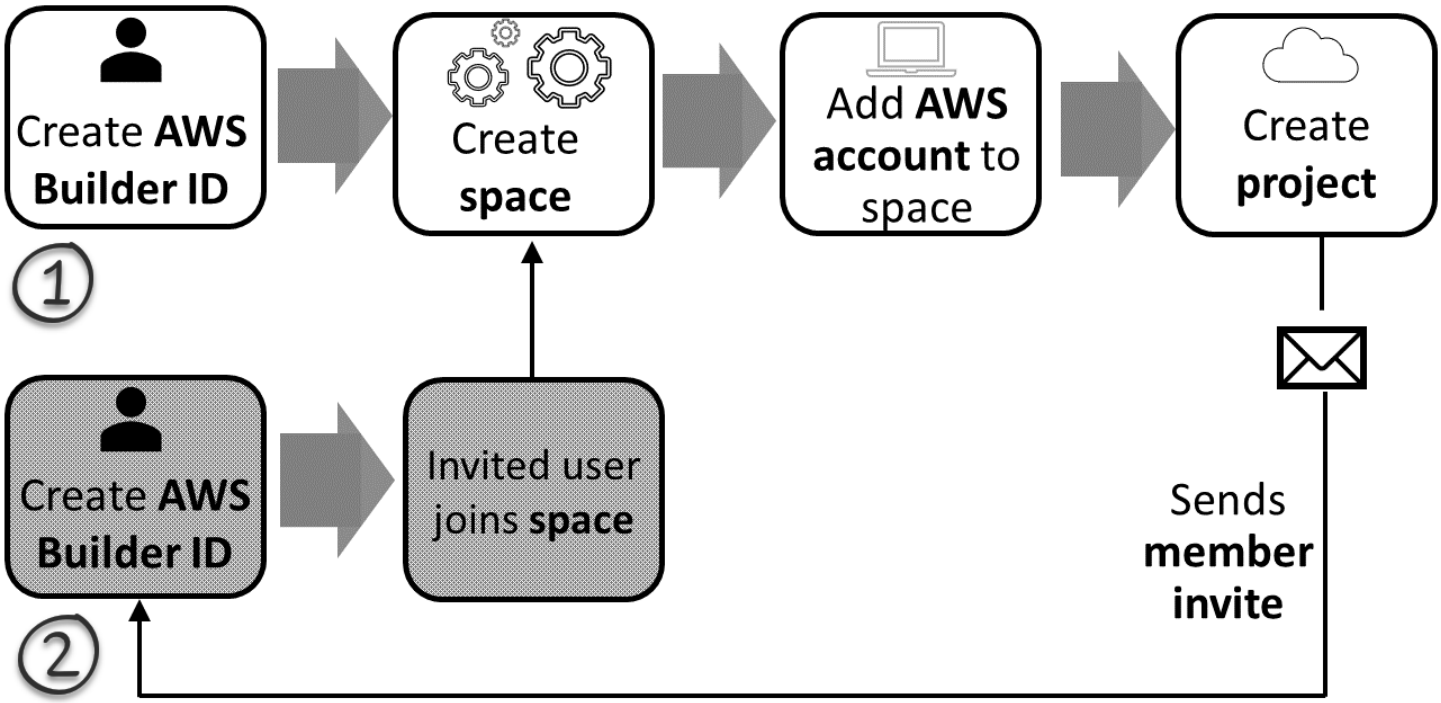
AWS Builder ID スペースの設定と管理の手順は、このガイドに記載されています。 CodeCatalyst AWS Builder ID スペースを使用するには、サインインに使用したユーザー設定と AWS Builder ID CodeCatalyst を使用してセットアップします。 CodeCatalyst

ID フェデレーションをサポートするスペースの設定と管理の手順は、『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、 AWS Builder ID スペースを使用して Amazon CodeCatalyst で作業するための一般的な設定方法として、スペースとプロジェクトを最初のユーザーとして作成する方法と、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方法の 2 つを紹介します。これらの設定ワークフローは必然的に全く異なります。以下の図は、両方のサインアッププロセスを以下のように示しています。

Amazon CodeCatalyst で作業するための設定には、2 つの一般的な方法があります。1 つは最初のユーザーとしてスペースとプロジェクトを作成する方法、もう 1 つは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。これらの設定ワークフローは必然的にまったく異なります。以下の図は、両方のサインアッププロセスを以下のように示しています。

1. 前者の場合は、会社、チーム、またはグループ用のスペースを作成して設定し、他のユーザーをこれらのリソースに招待する前にプロジェクトを作成します。 AWS アカウント 請求目的で提供する必要がありますが、無料利用枠はデフォルトでもかまいません。
2. 後者の場合、 CodeCatalyst プロジェクトへの招待を承諾して参加した場合、他の誰かがあなたのためにスペースとプロジェクトをすでに作成しています。ただし、他のユーザーと一緒に作業を開始できるよう、プロフィールを設定しておく必要があります。



> **Tip**
>
> CodeCatalyst 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。初めてサインアップすると CodeCatalyst、プロジェクトだけでなくスペースも作成するように求められます。

スペースとプロジェクトを作成するためにサインアップする場合でも、招待状を受け取る際にサインアップする場合でも、 AWS ログインに使用するビルダー 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、および電子メールアドレスを指定します。 CodeCatalyst この時点以降は、E メールとパスワードを使用してサインインします。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。

AWS Builder ID では CodeCatalyst 、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 CodeCatalyst プロジェクトの言語設定と通知設定に関する設定が含まれます。

> **Tip**
>
> Amazon CodeCatalyst プロフィー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。さらに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

サインアップして最初のスペースと開発ロールを作成する

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、 AWS Builder ID 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は、次のようなフローがあります。

メアリー・メジャーは、 に関心があり CodeCatalyst 、それを試すデベロッパーです。CodeCatalyst コンソールに移動し、サインアップして AWS Builder ID を作成するオプションを

選択します。Mary は 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。Builder ID AWS を使用して、 CodeCatalyst やその他のアプリケーションにサインインできます。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確認できるように、作成する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。メアリー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は Modern 3 層ウェブアプリケーションのブループリントを試すことにしました。必須フィールドを入力し、プロジェクトを作成します。プロジェクトの準備が整うとすぐに、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。表示される内容に興味があれば、同僚の一部をプロジェクトに招待して、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 を使用して にサインインするように AWS Builder ID を設定します。MFA を設定すると、Mary は承認されたサードパーティー認証アプリの CodeCatalyst パスワードとパスコードまたはトークンの組み合わせ CodeCatalyst を使用して 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、スペースに アカウント、サポートロール、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、 CodeCatalyst ワークフローが AWS AWSリソースにアクセスできるようにする 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AWS のサービスであり、サインインしたアカウントで作成されます。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、管理アクセスは、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デベロッパーにのみ割り当てます。

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 を指定する準備が必要です。12 桁の AWS アカウント ID を準備します。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。AWS Builder ID 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. 名前に、 AWS Builder ID を使用するアプリケーションに表示する姓名を入力します。スペースを使用できます。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、スパムフォルダまたは迷惑メールフォルダに見つからない場合は、コードを再送信する 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. CodeCatalyst 「エイリアスの作成」ページで、 の一意のユーザー識別子に使用するエイリアスを入力します CodeCatalyst。など、スペースを含まない短縮バージョンの名前を選択しますMaryMajor。 CodeCatalyst 他のユーザーは、コメントやプルリクエストでこれを

@mention に使用します。 CodeCatalyst プロファイルには、 AWS Builder ID 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 を行います。

[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、 を追加するための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、サインインが必要になる場合があります。

    でAWS Management Console、スペースを作成するAWS リージョンのと同じ 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/. にあるの 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）「承認された有料階層」で、「有料階層 (標準、エンタープライズ) の承認」を選択し、請求アカウントの有料階層をオンにします。

> **Note**
>
> これにより、請求枠が有料枠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 が設定されAWS アカウント、 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と、スペースは無料利用枠のみ使用できます。

16. [スペースを確認] を選択します。

    アカウント検証成功メッセージが表示され、アカウントがスペースに追加されたことが示されます。

17. 「Amazon CodeCatalyst スペースの検証」ページに残ります。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。スペースの準備が整いました。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。

    - スキップを選択して今すぐ にします。
    - 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 を開始する前に、 を開きAWS Management Console、スペース AWS アカウントに対して同じ 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. ロールを作成する AWS アカウント 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AWS Management Consoleを選択します。

   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 開発管理者ロールを作成するを選択します。このオプションでは、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し、 AdministratorAccessAWSマネージド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する場合にのみ推奨されます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 CodeCatalystで、アカウントに追加された IAM CodeCatalystWorkflowDevelopmentRole-*spaceName*ロールのリストに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 を開始する前に、 を開きAWS Management Console、スペース AWS アカウントに対して同じ 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. ロールを作成する AWS アカウント 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AWS Management Consoleを選択します。

   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、サインインが必要になる場合があります。
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。このオプションでは、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、AWSRoleForCodeCatalystSupport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にし、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`*ロールのリストに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。

AWS Builder ID を作成し、最初のスペースを作成して、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、「Amazon でのプロジェクトの作成 CodeCatalyst」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。

AWS招待の承諾とビルダー ID の作成

CodeCatalyst 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、AWSビルダー ID を作成するよう求められます。AWSビルダー ID CodeCatalyst を使用して内のリソースにアクセスします。

> **Tip**
>
> さらに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。

以下は、 CodeCatalyst ユーザーがプロジェクトまたはスペースへの招待から始める場合に考えられるフローの 1 つです。

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よう招待された開発者です。Saanvi が招待を承諾すると、のサインインページが開きます。 CodeCatalyst彼女はサインアップを選択し、電子メールアドレスとパスワードを入力してビルダー ID を作成します。AWSSaanvi は AWS Builder ID を使って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 CodeCatalyst 。その後、プロフィールを編集してログイン用のメールアドレスやパスワードを変更できます。エイリアスの選択を求められると、Saanvi は @mention Saanvi CodeCatalyst に表示され、`SaanviSarkar` CodeCatalyst 他のプロジェクトメンバーもそのエイリアスを使用するエイリアスとして指定します。サインアップすると、Saanvi は自分のサインイン認証情報を Builder ID 認証情報を使用する他のアプリケーションでも使用できるようになります。AWS

サインアップが完了すると、Saanvi CodeCatalyst は招待状で指定されたプロジェクトとスペースに自動的に参加します。招待状には、プロジェクトとスペースでの彼女の役割に対するあらかじめ決められた権限も与えられます。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とともに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、CodeCatalyst 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際の認証に使用されます。

Saanvi がプロジェクトに取り組むと、Saanvi のエイリアスはそのプロジェクトの作業アクティビティログに記録されます。Saanvi が投稿した課題やコメントにはエイリアスが表示され、他のプロジェクトメンバーは Saanvi からの返信で @mention を送ることができます。別のプロジェクトメンバーの @mention に、Saanvi はプロフィールでそのエイリアスを調べます。 CodeCatalyst

時間があれば、Saanvi AWS はビルダー ID を多要素認証 (MFA) CodeCatalyst でサインインするように設定します。MFA を設定すると、Saanvi CodeCatalyst は自分のパスワードと、 CodeCatalyst 承認されたサードパーティ認証アプリのパスコードまたはトークンを組み合わせてサインインできます。

招待を受け入れ、ビルダー ID を作成する。AWS

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に求めるメールが届きます。AWSBuilder ID を既にお持ちでサインインしている場合 CodeCatalyst、「招待を承認」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、AWSビルダー ID を持っている場合は、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。

AWSBuilder ID をお持ちでない場合は、[招待を承認] を選択するとサインインページに移動します。ここで AWS Builder ID を作成するオプションを選択する必要があります。

AWS招待を受け入れてビルダー ID を作成するには

1. 招待メールで [招待を承認] を選択します。
2. サインインページで [サインアップしていませんか?] を選択します。AWSビルダー ID を作成します。

   > **Tip**
   >
   > AWSビルダー ID はサインインするために作成する ID です。とは同じではありません AWS アカウント。

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

   「名前」に、AWS Builder ID を使用するアプリケーションに表示したい名と姓を入力します。スペースは使用できます。AWSこれはビルダー ID のプロファイル名 (「Mary Major」など) 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。

   指定したメールアドレスに確認コードが送信されます。このコードを [認証コード] に入力し、[確認] を選択します。5 分経ってもコードが届かず、迷惑メールフォルダーにも見つからない場合は、[コードを再送信] を選択します。
4. 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。
5. 「AWSビルダー ID を作成」を選択します。
6. 「エイリアスの作成」ページで、固有のユーザー識別子として使用するエイリアスを入力します CodeCatalyst。たとえば、スペースを含まない短縮形の名前を選択してください。MaryMajor CodeCatalyst 他のユーザーはコメントやプルリクエストでこれを使ってあなたを @mention します。 CodeCatalyst プロフィールには、AWS Builder ID CodeCatalyst のフルネームと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、の別の場所に表示されます CodeCatalyst。たとえば、アクティビティフィードにはリストされているアクティビティのプロフィール名が表示されますが、プロジェクトメンバーは @mention youのエイリアスを使用します。

[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 でサインインするためのメール招待を承諾する

ID フェデレーションをサポートするスペースに追加されたユーザーには、サインインポータルのリンクと設定情報が記載されたメールが届きます。以下の手順に従って招待を承諾し、SSO でサインインします。

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 を受け入れてサインインするには

1. メールに記載されている、リクエストを受け入れるボタンを選択します。メールに記載されているリンクを使用して、会社に関連するスペースのサインインポータルに移動します。
2. [ユーザー名] と [パスワード] に、認証情報を入力します。
3. [Sign in (サインイン)] を選択します。

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。

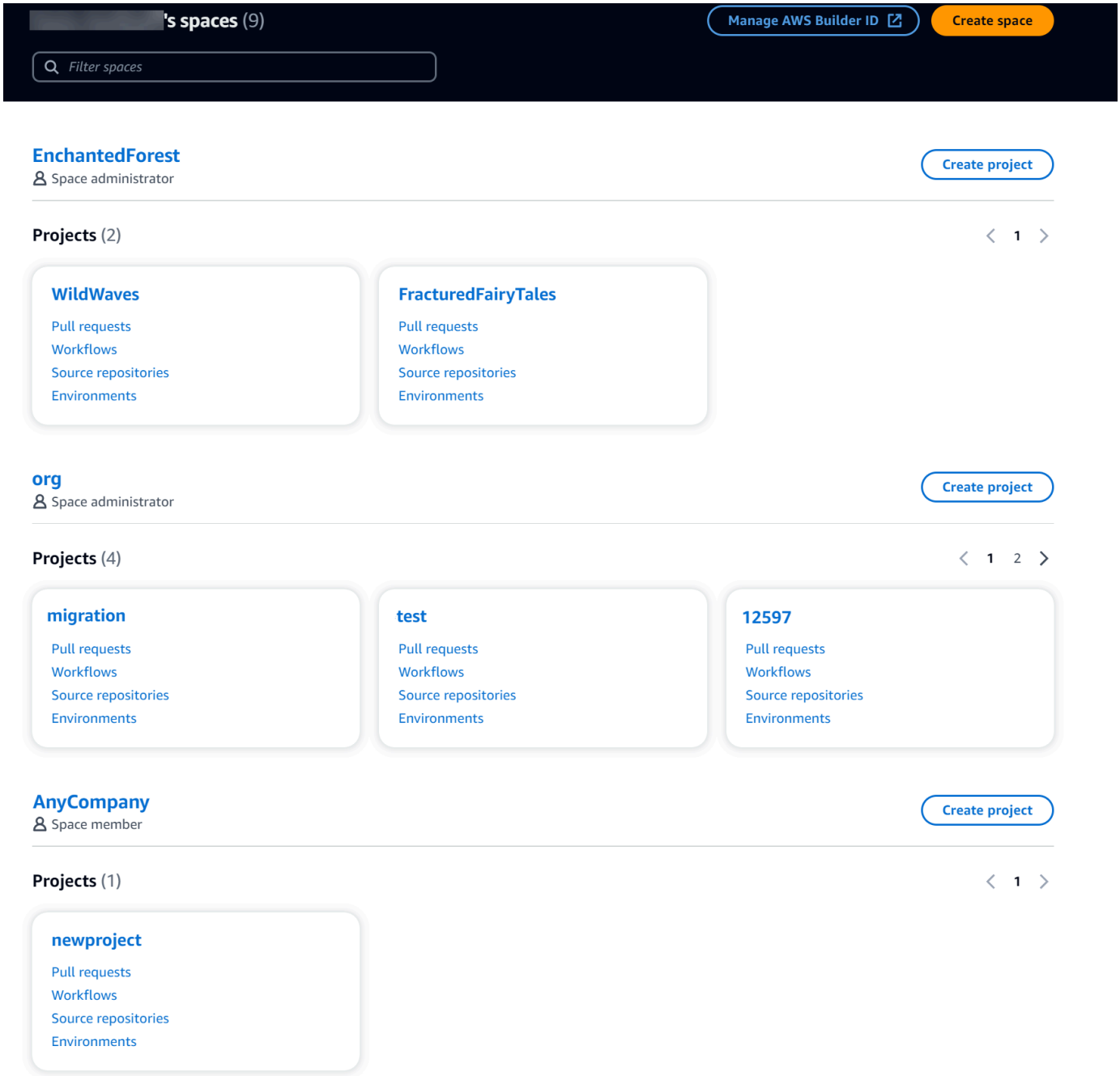
SSO でサインインするには

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、「Amazon での個人アクセストークンの管理 CodeCatalyst」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。

- Amazon での個人アクセストークンの管理 CodeCatalyst

- ログに記録されたイベントへのアクセス CodeCatalyst

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、「セットアップ 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:コンポーザブル 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シングルページアプリケーションブループリント入門
- でのワークフロー入門 CodeCatalyst
- 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 へのアーティファクトのアップロード

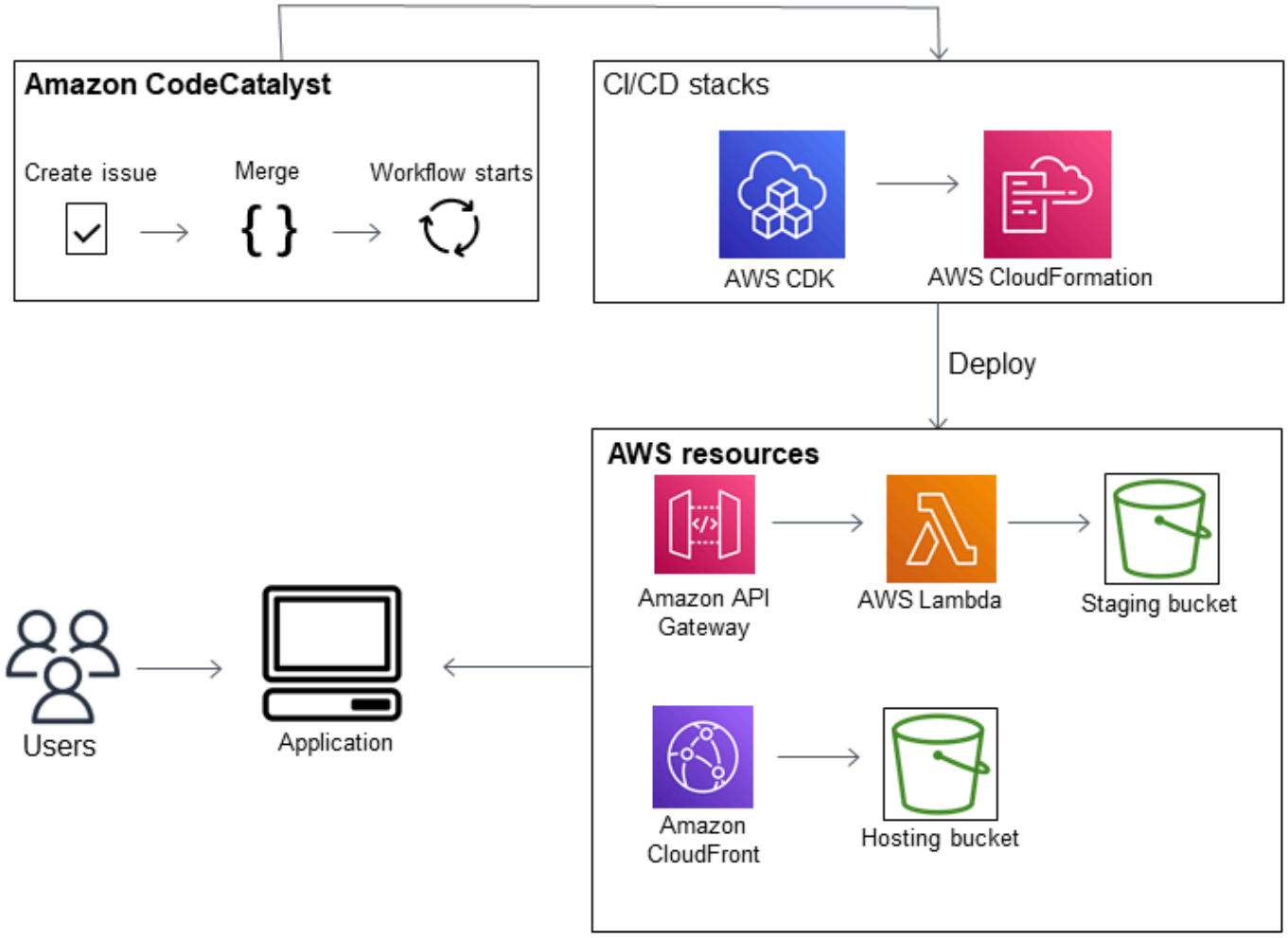
- チュートリアル:を使用してサーバーレスアプリケーションをデプロイする AWS CloudFormation
- チュートリアル:Amazon ECS へのアプリケーションのデプロイ
- チュートリアル:Amazon EKS へのアプリケーションのデプロイ
- チュートリアル:アクションを使った lint コード GitHub
- チュートリアル:React アプリケーションの作成と更新

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。プロジェクトブループリントリファレンス

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。セットアップ 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、AWS Builder ID ユーザーをサポートするスペースの作成、スペースユーザーの管理、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用の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、「AWS アカウント スペースの管理」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、「の問題点 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、「開発環境では統合開発環境がサポートされています。」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のワークフローを使用したテスト CodeCatalyst。

   > **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、「の問題点 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セットアップ 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、「ソースリポジトリをリンクする」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、「開発環境では統合開発環境がサポートされています。」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、「のワークフローによるビルド、テスト、デプロイ CodeCatalyst」を参照してください。

でのワークフロー入門 CodeCatalyst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、「の問題点 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

Amazon の生成 AI 機能はプレビューリリース CodeCatalyst であり、変更される可能性があります。これらは、米国西部 (オレゴン) リージョンでのみ使用できます。生成 AI 機能へのアクセスは、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。

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、タスクを達成するために時間よりも多くの作業を行います。他のユーザーが一見して変更を見つけることを期待して、それらの変更をレビューするためのプルリクエストを作成するときに、チームメートにコードの変更を説明する時間はほとんどありません。また、プルリクエストの作成者とレビュー担当者には、プルリクエストに関するすべてのコメントをよく検索して読み取る時間がありません。特にプルリクエストに複数のリビジョンがある場合、 には生成 AI 機能 CodeCatalyst が含まれており、どちらもチームメンバーによるタスクの迅速な達成と、作業の最も重要な部分に集中する時間の増加に役立ちます。

> **Note**
>
> Amazon Bedrock を搭載: AWS 自動不正使用検出を実装します。Amazon Bedrock には、「Write description for me」、「Create content summary」、「Assign issue to Amazon Q feature development capabilities」が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、人工知能 (AI) の安全性、セキュリティ、責任ある使用を実現できます。

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、簡単なコード変更や改善のための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。

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 によって初期ソリューションが作成されるよう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、「AWS Builder ID ユーザーをサポートするスペースの作成」、「の問題点 CodeCatalyst」、「の拡張機能 CodeCatalyst」、および「Amazon のロールを使った作業 CodeCatalyst」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の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。別の設計図で作成されたプロジェクトを使用する場合でも、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わずかな変更や修正、主要な機能の追加、新しいバージョンのコード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意図を要約することは、コードを確認する他のユーザーにとって役立ち、時間の経過に伴うコードの変更履歴を理解するのにも役立ちます。ただし、開発者は、レビュー担当者がレビュー対象やコードの変更の背後にある意図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、コードを使用して自分自身を説明したり、あいまいな詳細を提供したりすることがよくあります。

プルリクエストを作成するときに Write description for me 機能を使用して、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ことができます。このオプションを選択すると、Amazon Q はコード変更を含む送信元ブランチと、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、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のブループリント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**
>
> 別の設計図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、機能をテストするための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クイックコード変更を加えます。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、「追加」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、Create 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 5 **3**から 5 に変更します。

> **Tip**
>
> 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、integ 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。

変更のある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、追加 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を行うには、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. 送信元ブランチ で、test_endpoint.py先ほど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、説明の書き込み を選択すると、プルリクエストに含まれる変更の説明が に CodeCatalyst 作成されます。
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、同意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よく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は、レビュー担当者を追加するか、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関す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が残されることがよくあります。多くのレビュー担当者からのコメントが多数ある場合は、フィードバックで一般的なテーマを選択することや、すべてのリビジョンのすべてのコメントを確認したことを確認するのが困難な場合があります。コメント概要の作成機能を使用すると、プルリクエストのコード変更に関するすべてのコメントを Amazon Q で分析し、それらのコメントの概要を作成できます。

> **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なく、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。

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、コードベースの特定の部分を調査する必要があること、またはその他の緊急作業を最初に調査する必要があるため、問題が解決しないことがあります。 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ながら、お客様やお客様のチームは注意が必要な問題に集中して作業を最適化できます。

> **Tip**
>
> Amazon Q は、単純な問題と単純な問題に最も効果的です。最良の結果を得るには、プレーン言語を使用して、何をするか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 で問題をどのように処理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に返信して、必要に応じてそのアプローチを繰り返し処理できます。このオプションを選択した場合、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。各ステップの確認を選択しない場合、Amazon Q は作業をより迅速に完了する可能性がありますが、問題または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。
- 作業の一環としてワークフローファイルの更新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設定されたワークフローがある場合があります。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内に存在しないことが確実でない限り、Amazon Q がワークフローファイルで作業できるように選択しないでください。
- 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に対してのみ作業できます。リンクされたリポジトリはサポートされていません。

選択内容を確認してから、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に基づいてい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、作業のステータスの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、作業の正確な時系列レコードは保持されません。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が表示されます。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、問題をレビュー中の に移動し、評価できるコードがあることをユーザーとユーザーのチームが認識できるようにします。

> **Note**
>
> この機能は、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た場合、問題は状態を変更しない可能性があります。最良の結果を得るには、標準ボードレイアウトを持つプロジェクトでのみこの機能を使用してください。
>
> 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。一度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。

チュートリアルのこの部分では、 Modern の 3 層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 つはワークフローを更新して という名前のブランチを含めるためのものです`test`。

> **Note**
>
> 異なるコードの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、わかりやすい言葉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、タイトルを と入力します**Create another mysfit named Quokkapus**。説明 で、次の詳細を入力します。

```
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) 問題への mysfit のサムネイルとプロファイル画像として使用する画像を添付します。これを行う場合は、説明を更新して、使用するイメージの詳細とその理由を含めます。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。これらのイメージをソースリポジトリに追加してください。」

> **Note**
>
> このチュートリアルでは、アタッチされたイメージはウェブサイトにデプロイされません。イメージを自分でウェブサイトに追加し、プルリクエストを作成した後に使用するイメージを指すように Amazon Q にコメントを残すことができます。

説明を確認し、次のステップに進む前に、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. 各ステップの後に Amazon Q を停止するように要求 を選択し、その作業セレクタがアクティブ状態になるまでレビューを待ちます。

> **Note**
>
> すべてのステップの後に Amazon Q を停止するオプションを選択すると、問題にコメントし、コメントに基づいて Amazon Q にアプローチを 3 回まで変更させることができます。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すると、Amazon Q がフィードバックに待機していないため、作業がより迅速に進行する可能性がありますが、コメントを残すことで Amazon Q の方向性に影響を与えることはできません。こ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に変更することを許可する。
9. 問題の作成を選択します。ビューが Issues ボードに変更されます。
10. 「問題の作成」を選択して別の問題を作成します。今回は、「」というタイトルのものを作成します**`Change the get_all_mysfits() API to return mysfits sorted by the Age attribute`**。この問題は Amazon Q に割り当てて、問題を作成します。
11. 「問題の作成」を選択して別の問題を作成します。今回は、「」というタイトルのものを作成します**`Update the OnPullRequest workflow to include a branch named test in its triggers`**。オプションで、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 が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てください。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、アットシンボル (@) を入力し、ファイル名を入力します。

問題を作成して割り当てると、問題は進行中の に移行します。Amazon Q は、ピン留めされたコメントで問題内の進行状況を追跡するコメントを追加します。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バックグラウンドセクションと、ソリューション作成の提案アプローチの詳細を含む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に対するソリューションの起動に成功すると、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。コードの準備ができたら、プルリクエストを作成します。これにより、提案されたコード変更を確認し、そのプルリクエストへのリンクを問題に追加し、問題をレビュー中 に移行できます。

> **Important**
>
> プルリクエストをマージする前に、プルリクエストのコード変更を必ず確認してください。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ず、マージ時にエラーが含まれている場合、コードベースコード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、Amazon Q に割り当てられた進行中の問題を選択します。コメントを確認して、ボット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ているアプローチを取り、X を選択して問題ペインを閉じます。
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、問題の説明に記録しているアプローチします。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。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、コードを確認します。
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、「プルリクエストのレビュー」を参照してください。Amazon Q で提案されているコードを変更する場合は、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指定してください。

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 の「ニーズ」と「a」の間にスペースを追加して、「説明を変更してタイプミス」を修正できます。 または、Amazon Q に説明を更新し、改訂され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、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で mysfit を更新できます。

> **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の各ステップの後に停止するという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) ユーザーや他のプロジェクトユーザーがコードの変更に必要なすべてのコメントを残したら、リビジョンの作成を選択して、コメントで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 Amazon Q に作成させます。リビジョン作成の進行状況は、変更ではなく、概要 で Amazon Q によって報告されます。ブラウザを更新して、リビジョンの作成時に Amazon Q 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。また、マージする前に、追加のワークフローと環境を作成してコードをテストすることもできます。詳細については、「でのワークフロー入門 CodeCatalyst」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、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、このチュートリアル中に作成した不要になっ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。

- 今後対処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場合、またはソリューションが見つからない場合は、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「料金表」を参照してください。
- 作業が完了した問題がある場合は、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:コンポーザブル 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 では、プロジェクトをすばやく開始できるよう、さまざまな設計図を用意しています。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、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、ビルドおよびテストレポート、統合問題追跡ツールなど、必要なリソースが含まれます。ただし、プロジェクトを徐々に構築したり、ブループリントで作成された既存のプロジェクトに機能を追加したりしたい場合もあります。ブループリントを使ってこれを行うこともできます。このチュートリアルでは、基礎を築き、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から始める方法を紹介します。そこから、都合の良いときに最初のブループリントに加えて他のブループリントを適用することで、追加のリソースや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またが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 プロジェクト開発キット (AWS PDK) ブループリントをまとめて、React ウェブサイト、Smithy API、および 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 には、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が用意されています。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。

以下の PDK ブループリントは、相互に使用してコンポーザブルな方法でアプリケーションを構築できるように設計されています。

- Monorepo-monorepo 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可視化も行います。

- タイプセーフ API-Smithy または OpenAPI v3 のいずれかで定義できる 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 を実装したり操作したり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 コンストラクトをベンド。
- Cloudscape React ウェブサイト-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。これには Cognito Auth と、（オプションで）作成した API が事前に統合されています。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 CDK 関連のインフラストラクチャーをすべてセットアップ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 プロジェクト開発キット (AWS PDK) DevOps に含まれる構成と互換性のある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する方法、他のユーザーをそのアプリケーションに招待する方法、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。プルリクエストは、プルリクエストがマージされたときに、接続されている AWS アカウントのリソースに自動的にビルドおよびデプロイされます。

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、プロジェクト内の次のリソースを使用してプロジェクトが作成されます。 CodeCatalyst

- モノリポジトリとして設定されたソースリポジトリ。
- デフォルトブランチに変更が加えられるたびに、静的コード分析とライセンスチェックを実行するほか、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。コードに変更を加えるたびに、アーキテクチャ図が生成されます。
- 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。
- 自動レポート機能を備えたテストレポートスイート。

トピック

- 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
- ステップ 5: DevOps プロジェクトをデプロイするためのワークフローを設定する
- ステップ 6: リリースワークフローを確認し、ウェブサイトを表示する
- PDK プロジェクトのコラボレーションと繰り返し

前提条件

プロジェク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。セットアップ CodeCatalyst

- AWS サインインするためのビルダー ID CodeCatalyst 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、AWS Builder ID ユーザーをサポートするスペースの作成、スペースユーザーの管理、およびスペース管理者ロールを参照してください。
- スペースに関連付けられた AWS アカウントと、サインアップ時に作成した IAM ロールを持っていること。たとえば、サインアップ時に 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ことを選択できます。`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。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用の CodeCatalystWorkflowDevelopmentRole-***spaceName*** ロールの作成。

ステップ 1: monorepo 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、基盤となるモノレポコードベースを作成します。これにより、PDK ブループリントをさらに追加できるようになります。

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/ でコンソールを開きます。 CodeCatalyst
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. 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. [設計図から始める] を選択します。

5. PDK-Monorepo ブループリントを選択し、[次へ] を選択します。
6. 「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と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. 「主要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。Java または Python TypeScript から選択できます。

   b. [コード設定] を選択します。

   c. 「ソースリポジトリ」テキスト入力フィールドに、新しいリポジトリを作成するソースリポジトリの名前を入力するか、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なければなりません。詳細については、「ソースリポジトリをリンクする」を参照してください。

   d. (オプション) Package マネージャー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、 TypeScript プライマリプログラミング言語として選択した場合にのみ必要です。
8. (オプション) 選択したプロジェクトパラメータ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を生成」から「コードを表示」を選択します。
9. (オプション) ブループリントのカードから [詳細を表示] を選択すると、ブループリントのアーキテクチャの概要、必要な接続と権限、ブループリントが作成するリソースの種類など、ブループリントに関する特定の詳細が表示されます。
10. [プロジェクトの作成] を選択して monorepo プロジェクトを作成します。作成したルートレベルのプロジェクトは、monorepo 内のプロジェクト間の相互依存関係を管理し、ビルドのキャッシュと依存関係の管理も行います。

    プロジェクトブループリントの詳細については、を参照してください。プロジェクトブループリントリファレンス

PDK-Monorepo ブループリントはプロジェクトの基盤のみを生成します。ブループリントを使用して実行可能なアプリケーションを作成するには、Type Safe API、Cloudscape React Web サイト、インフラストラクチャ、またはなど、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、タイプセーフ API をプロジェクトに適用します。

ステップ 2: タイプセーフ API をプロジェクトに追加する

PDK-タイプセーフ API ブループリントを使用すると、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コードが含まれます。ブループリントでは、すべての API 操作に対して型安全性のある CDK コンストラクトも生成されます。ブループリントを既存の PDK monorepo 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-Type Safe API ブループリントを適用するには

1. monorepo 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-Type Safe API ブループリントを選択し、「次へ」を選択します。
3. 「ブループリントの設定」で、ブループリントのパラメータを設定します。

   - [モデル言語] で、API モデルを定義す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、API をデプロイする CDK インフラストラクチャーの作成に使用する言語を選択します。
   - 「ハンドラー言語 (s)」ドロップダウンメニューを選択し、API オペレーション用のハンドラーを実装する言語を選択します。
   - 「ドキュメンテーションフォーマット」ドロップダウンメニューを選択し、API ドキュメントを生成するのに必要なフォーマットを選択します。
4. 「コード変更」タブで、提案されている変更内容を確認します。プルリクエストに表示される差異は、プルリクエストが作成された時点でのプロジェクトへの変更を示しています。
5. ブループリントが適用されたときに加えられる変更案に満足したら、[ブループリントを適用] を選択します。

   プルリクエストが作成されたら、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。ファイルなどのリソースへのリンクを追加するには、@記号に続けてファイル名を指定します。

   > **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、「プルリクエストのレビュー」および「プルリクエストをマージする」を参照してください。

6. 「ステータス」列から「PDK-Type Safe API ブループリント」行の「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のリンクを選択します。

7. [Merge] を選択し、希望のマージ戦略を選択してから [Merge] を選択すると、適用したブループリントの変更が反映されます。

   プルリクエストがマージされると、monorepo `packages/apis/mypdkapi` プロジェクト内に新しいフォルダーが生成されます。このフォルダーには、設定した Type Safe API の API 関連のソースコードがすべて含まれています。

8. ナビゲーションペインで [ブループリント] を選択し、PDK-Type Safe API のステータスが Up to date になっていることを確認します。

ステップ 3: プロジェクト用に Cloudscape React ウェブサイトを追加します。

PDK-Cloudscape React ウェブサイトブループリントはウェブサイトを生成します。オプションのパラメーター (Type Safe API) を関連付けると、認証済みのタイプセーフクライアント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。また、インタラクティブな API エクスプローラーを使用してさまざまな API をテストすることもできます。

PDK-Cloudscape React ウェブサイトブループリントを適用するには

1. monorepo 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 ウェブサイトブループリントを選択し、「次へ」を選択します。
3. 「ブループリントの設定」で、ブループリントのパラメータを設定します。

   - Web サイト名のテキスト入力フィールドに、Web サイトの名前を入力します。
   - 「Type Safe API」ドロップダウンメニューを選択し、Web サイトに統合したい 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、提案されている変更を確認します。プルリクエストに表示される差異は、プルリクエストが作成された時点でのプロジェクトへの変更を示しています。
5. ブループリントが適用されたときに加えられる変更案に満足したら、[ブループリントを適用] を選択します。

   プルリクエストが作成されたら、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。ファイルなどのリソースへのリンクを追加するには、@記号に続けてファイル名を指定します。

> **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、「プルリクエストのレビュー」および「プルリクエストをマージする」を参照してください。

6. ステータス列から PDK-Cloudscape React Website ブループリント行の「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のリンクを選択します。
7. [Merge] を選択し、希望のマージ戦略を選択してから [Merge] を選択して、適用したブループリント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、monorepo `packages/websites/my-website-name` プロジェクト内に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フォルダーが生成されます。
8. ナビゲーションペインで「ブループリント」を選択し、PDK-Cloudscape React Web サイトのステータスに「最新版」と表示されていることを確認します。

次に、PDK-インフラストラクチャブループリントを適用して、ウェブサイトを 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ブループリントは、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設定します。また、デフォルトでダイアグラムを生成し、プロトタイピング用ナグパックに準拠させることができます。

PDK-インフラストラクチャー設計図を適用するには

1. monorepo 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ーブループリントを選択し、「次へ」を選択します。
3. 「ブループリントの設定」で、ブループリントの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ーの開発に使用する言語を選択します。

- Stack name テキスト入力フィールドに、 CloudFormation ブループリント用に生成されたスタックの名前を入力します。

> **Note**
>
> DevOps ワークフローを設定する次のステップに備えて、このスタックの名前を書き留めておいてください。

- Type Safe API ドロップダウンメニューを選択し、Web サイトに統合したい API ブループリントを選択します。
- 「Cloudscape React TS ウェブサイト」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ー内にデプロイしたいウェブサイトブループリント (たとえば、PDK-Cloudscape React ウェブサイト) を選択します。

4. 「コード変更」タブで、提案されている変更を確認します。プルリクエストに表示される差異は、プルリクエストが作成された時点でのプロジェクトへの変更を示しています。
5. ブループリントが適用されたときに加えられる変更案に満足したら、[ブループリントを適用] を選択します。

   プルリクエストが作成されたら、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。ファイルなどのリソースへのリンクを追加するには、@記号に続けてファイル名を指定します。

> **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、「プルリクエストのレビュー」および「プルリクエストをマージする」を参照してください。

6. ステータス列から PDK-Infrastructure ブループリント行の [保留中のプルリクエスト] を選択し、未解決の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、希望のマージ戦略を選択してから [マージ] を選択して、適用したブループリント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、monorepo `packages/infra` プロジェクト内に新しいフォルダが生成されます。このフォルダには、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、[PDK のステータス-インフラストラクチャ] が [最新] になってい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: DevOps プロジェクトをデプロイするための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 DevOps アカウントとロールを使用してプロジェクトをビルドおよびデプロイするために必要なワークフローを生成します。

PDK-ブループリントを適用するには DevOps

1. monorepo 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. 「ブループリントの設定」で、ブループリントの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、 CloudFormation デプロイする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」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 ブループリントで設定し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、「AWS アカウント スペースへの追加」を参照してください。
   - アプリケーションドロップダウンメニューのデプロイに使用するロールを選択し、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、`SourceArnProjectID`プロジェクト設定にある現在のロールに制限してください。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、monorepo プロジェクトをデプロイしたい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 サービス」を参照してください。

4. 「コード変更」タブで、提案されている変更を確認します。プルリクエストに表示される差異は、プルリクエストが作成された時点でのプロジェクトへの変更を示しています。
5. ブループリントが適用されたときに加えられる変更案に満足したら、[ブループリントを適用] を選択します。

   プルリクエストが作成されたら、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。ファイルなどのリソースへのリンクを追加するには、@記号に続けてファイル名を指定します。

   > **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、「プルリクエストのレビュー」および「プルリクエストをマージする」を参照してください。

6. ステータス列から PDK-Infrastructure ブループリント行の [保留中のプルリクエスト] を選択し、未解決の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、希望のマージ戦略を選択してから [マージ] を選択して、適用したブループリント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、monorepo `.codecatalyst/workflows` プロジェクト内に新しい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、PDK のステータスが Up to date になっていることを確認します。 DevOps

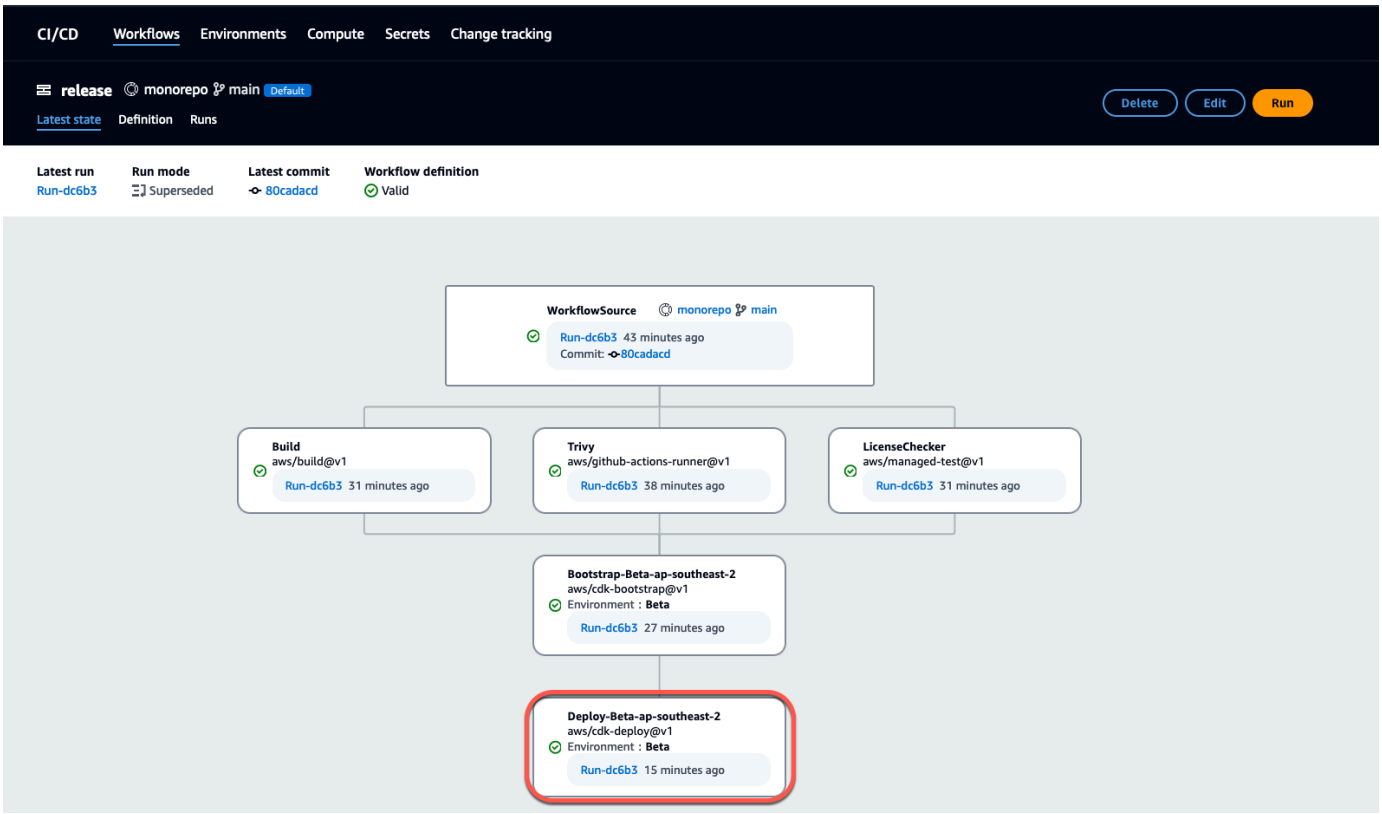
> **Note**
>
> 今後は PDK DevOps ブループリントとそれ以降の PDK ブループリントへのすべての変更が大幅に遅くなります。これは、バックグラウンドでロックファイルが生成され、ビルドやデプロイメントがfuture 繰り返せるようにするためです。サポート対象のすべての言語で、すべてのパッケージのロックファイルが生成されます。

ステップ 6: リリースワークフローを確認し、ウェブサイトを表示する

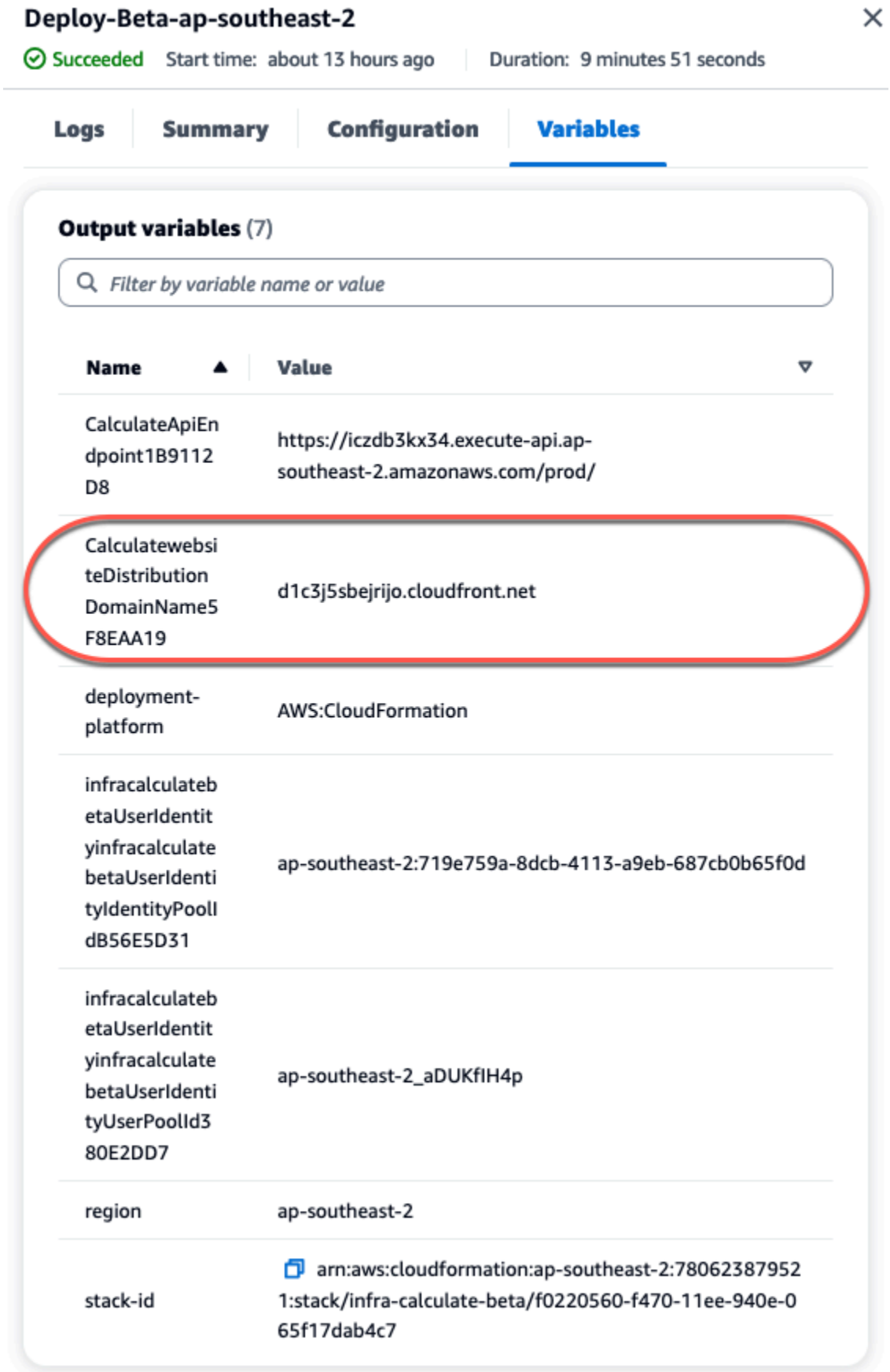
前のステップを完了したら、リリースワークフローを確認して、プロジェクトがビルドされていることを確認できます。

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-souteast -2 など) を選択し、[変数] を選択します。



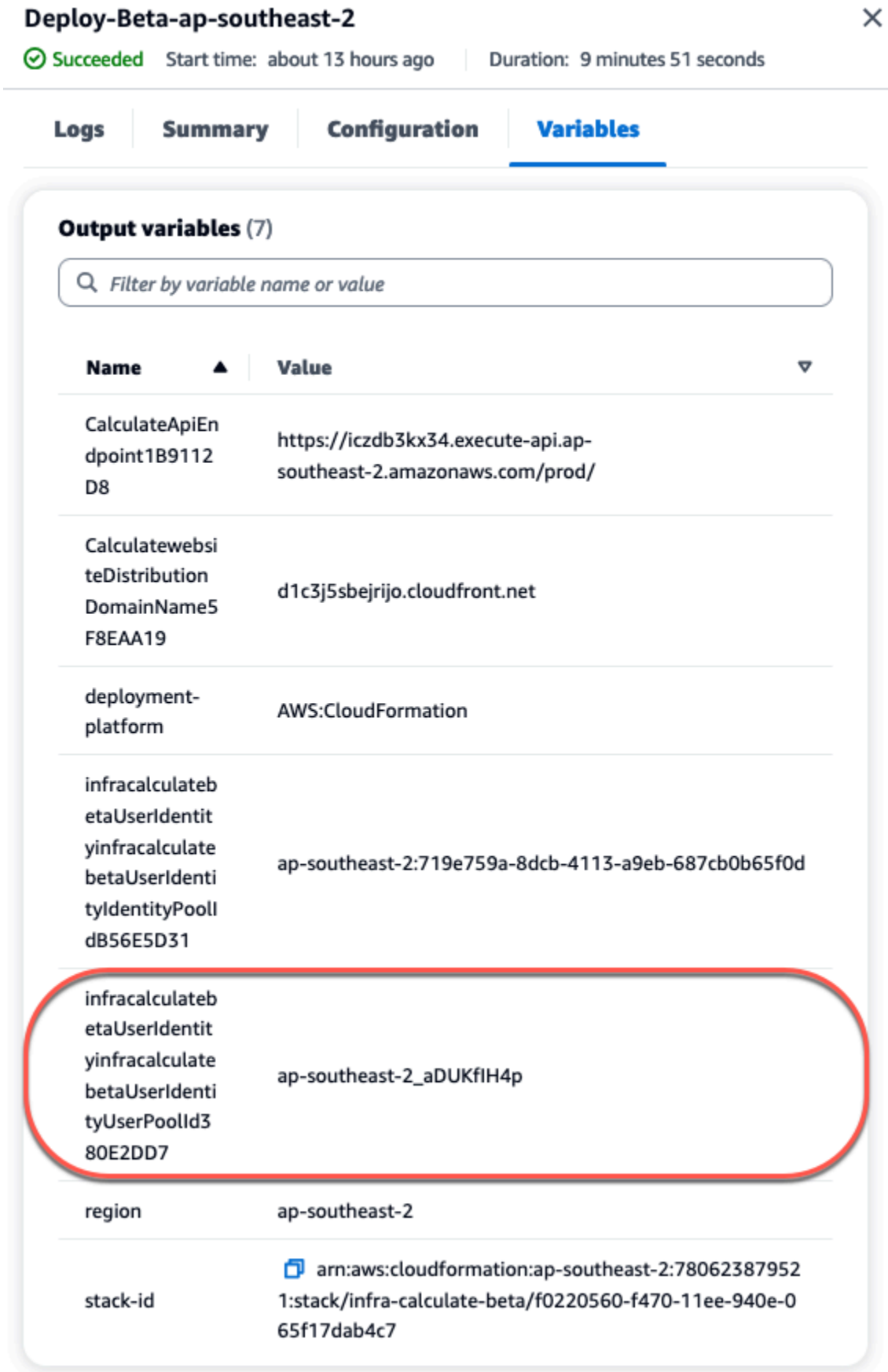
4. 変数テーブルにあるリンク (たとえば、***myPDKAPI*** websiteDistributionDomain NameXXXxx) をコピーして新しいブラウザウィンドウに貼り付けて、デプロイされた Web サイトを表示します。


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 ########################## ## (#:infra calculate XXXXX)#betaUserIdentityinfra* betaUserIdentity IdentityPoolId詳細については、「ユーザープール入門」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。

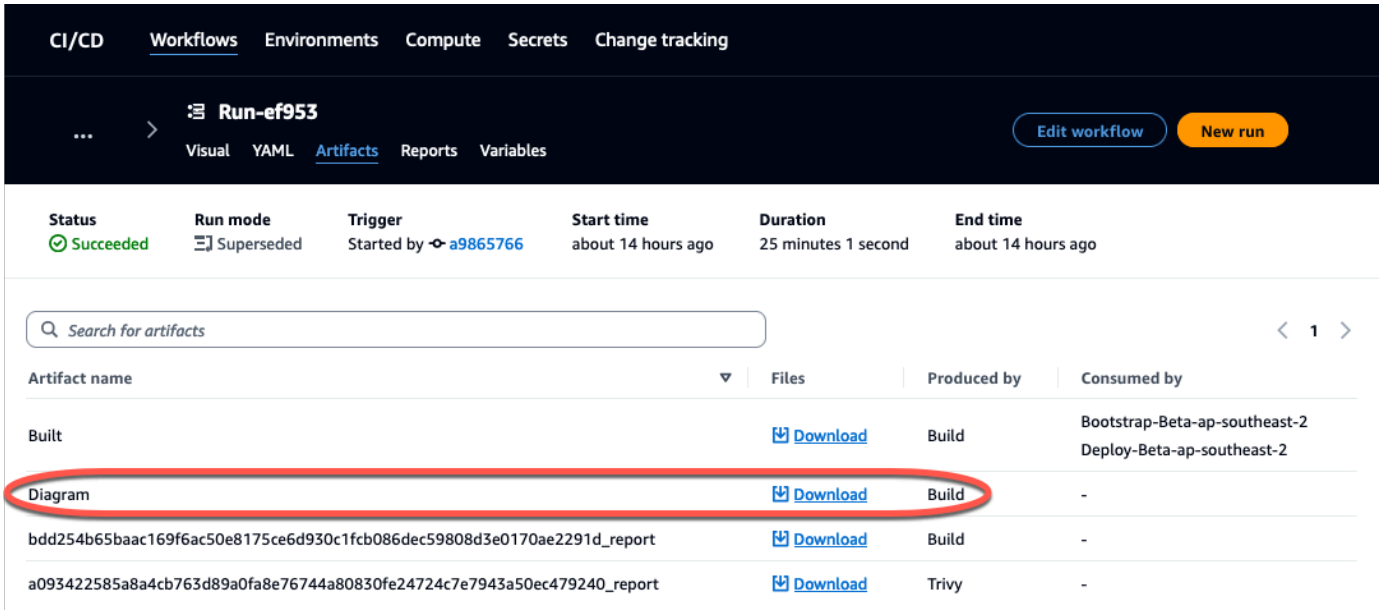
- 「招待メッセージ」で、「招待メールを送信」を選択します。
- 「ユーザー名」テキスト入力フィールドに、ユーザー名を入力します。
- 「メールアドレス」テキスト入力フィールドに、ユーザー名を入力します。
- 「仮パスワード」で「パスワードを生成」を選択します。

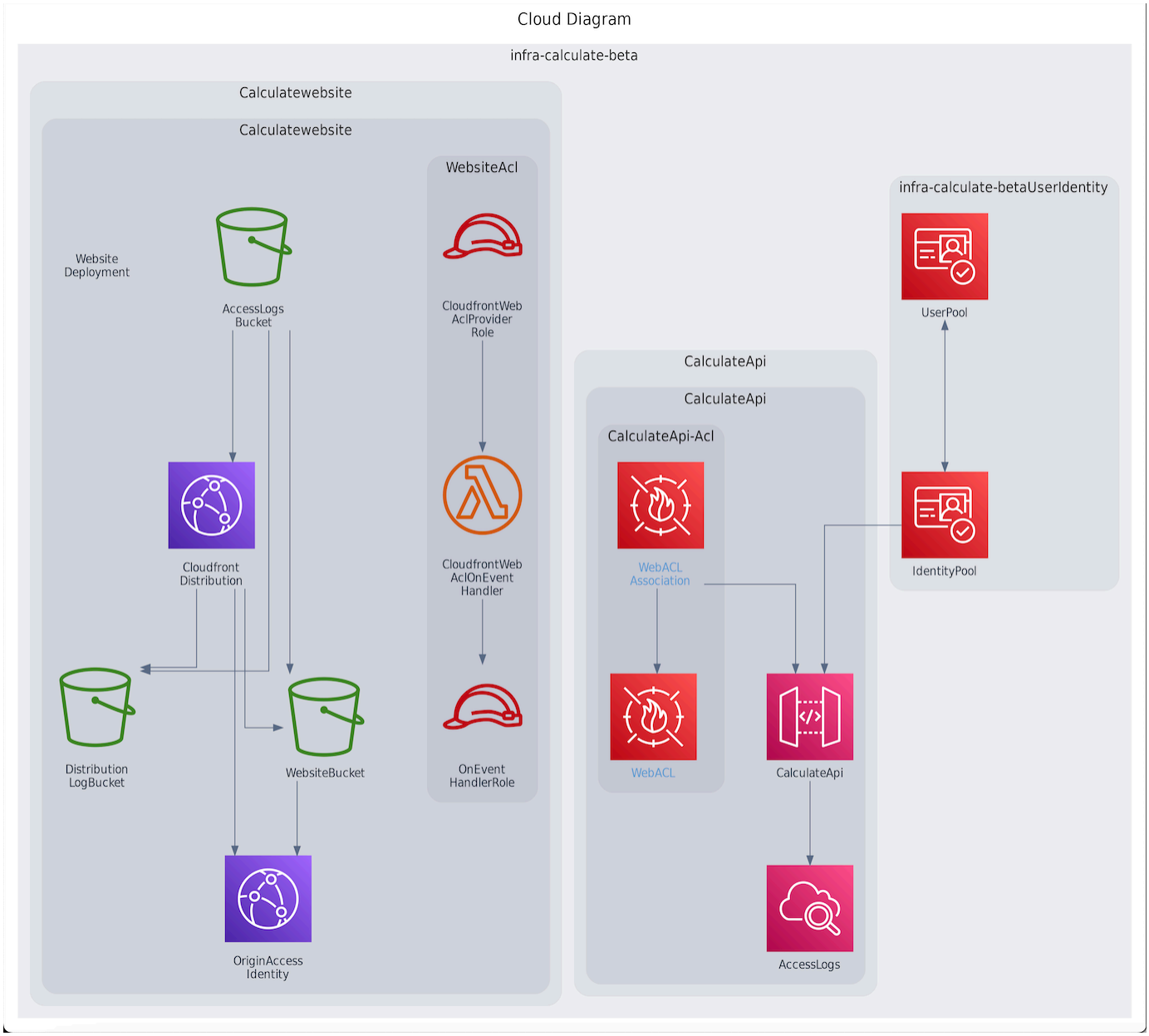
e. [ユーザーの作成] を選択します。

f. ユーザー情報パラメータに入力したメールアカウントに移動し、仮パスワードが記載されたメールを開きます。パスワードは書き留めておいてください。

g. デプロイした Web サイトに戻り、作成したユーザー名と受け取った仮パスワードを入力して、[Sign in] 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「アーティファクト」タブを選択し CodeCatalyst、「ダイアグラム」行で「ダウンロード」を選択して、ダウンロードしたファイルを開きます。





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、「Amazon のロールを使った作業 CodeCatalyst」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、「Amazon のロールを使った作業 CodeCatalyst」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、「の問題点 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、「開発環境では統合開発環境がサポートされています。」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、「プルリクエストをマージする」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、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、「プロジェクト内のブループリントの更新」を参照してください。

スペースイン 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 で作成したプロジェクト、メンバー、関連するクラウドリソースを追加するための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。削除したスペースの名前は再利用できません。

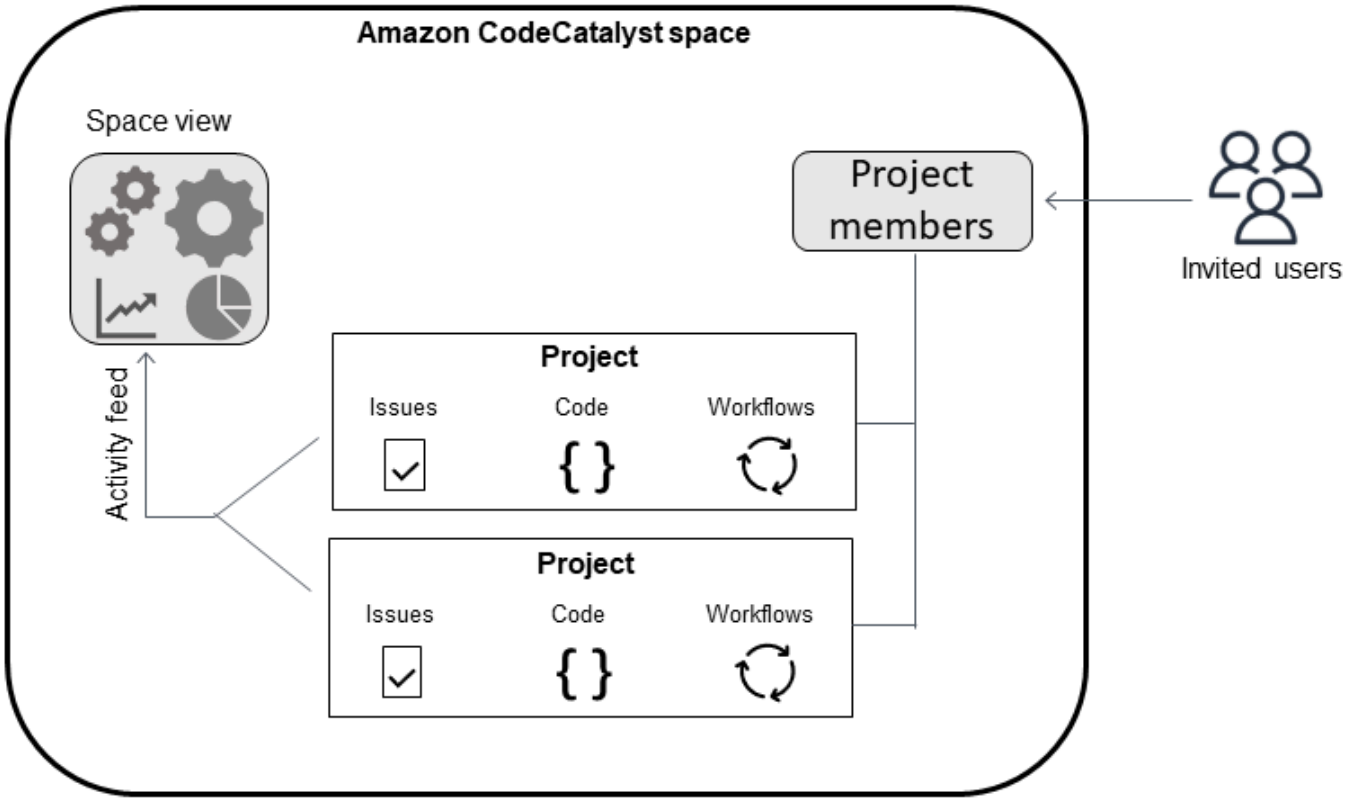
スペースを作成すると、自動的にスペース管理者ロールが割り当てられます。このロールはスペース内の他のユーザーに追加できます。

スペース管理者ロールでは、次のようにスペースを管理できます。

- 他のスペース管理者をスペースに追加します。
- メンバーの役割と権限を変更します。
- スペースを編集または削除する。
- プロジェクトを作成し、メンバーをプロジェクトに招待します。
- 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します。

スペースを作成すると、スペース管理者ロールと、スペース作成時に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 つのロールが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、そのユーザーが自動的にスペースにメンバーとして追加されます。このスペースのメンバーになっても、スペース内の権限は付与されません。スペースでユーザーができることは、特定のプロジェクトにおけるユーザーの役割によって決まります。

ロールの詳細については、「Amazon のロールを使った作業 CodeCatalyst」をご参照ください。



アカウントを追加する場合のその他の考慮事項は次のとおりです。

- one-to-one アカウント接続とスペースとのマッピングがあります。 AWS アカウント 1 AWS アカウント つのスペースを複数の異なるスペースに追加できます。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 スペースに追加すると、そのスペース内のどのプロジェクトでも使用できます。
- 各環境は複数をサポートできますが AWS アカウント、1 つのアクションで使用できるアカウントは 1 つの環境につき 1 つだけです。
- 請求はスペースレベルで設定されます。請求には複数のアカウントを設定できますが、1 CodeCatalyst つのスペースでアクティブにできるのは 1 つのみです。1 AWS アカウント つのスペースの請求アカウントとして使用できるのは 1 つだけです CodeCatalyst。アカウントをすでにスペースに使用している場合は、追加のスペースには別の請求先アカウントを使用する必要があります。
- ワークフローが環境内の AWS IAM ロールにアクセスする必要がある場合は、接続を作成した後、接続に IAM ロールを追加する必要があります。 CodeCatalyst 環境の使用方法の詳細については、「」を参照してください。環境を使用する

トピック

AWS Builder ID ユーザーをサポートするスペースの作成

AWS Builder ID を使用して Amazon CodeCatalyst 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、「セットアップ 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なければなりません CodeCatalyst。削除し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 CodeCatalyst ユーザーをサポートするスペースを作成するためのものです。ID フェデレーションをサポートするスペースの設定と管理の手順は、『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スペースをさらに作成するには、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しても、プロジェクトの作成を求めるメッセージは表示されません。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Amazon でのプロジェクトの作成 CodeCatalyst。

別のスペースを作成するには

1. で AWS Management Console、 AWS アカウント CodeCatalyst スペースに関連付けたいものでサインインしていることを確認します。
2. https://codecatalyst.aws/ CodeCatalyst で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なければなりません CodeCatalyst。削除し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、スペースに接続したいアカウントの 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 接続リクエストを承認する間は保存しておきたい場合もあります。

8. [認証する] を選択します。 AWS

9. に「Amazon CodeCatalyst スペースの確認」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。この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、 AWS リージョン スペースを作成したい場所と同じ場所を必ず選択してください。

   ページに直接アクセスするには、https://console.aws.amazon.com/codecatalyst/home/ AWS Management Console にある Amazon CodeCatalyst Spaces にサインインしてください。

   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成功メッセージが表示されます。

11. 「Amazon CodeCatalyst スペースの確認」ページを開いたままにします。次のリンクを選択してください: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 スペース詳細ページがで開きます。 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。

12. [利用可能な IAM ロール] CodeCatalyst で [IAM ロールを追加] を選択します。

    「利用可能な IAM ロールの追加」ページが表示されます。 CodeCatalyst

13. [IAM CodeCatalyst で開発管理者ロールを作成] を選択します。このオプションでは、開発ロールのアクセス権限ポリシーと信頼ポリシーを含むサービスロールが作成されます。

    開発者ロールは、 CodeCatalyst ワークフローが Amazon S3、Lambda、 AWS などの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`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](#)。

14. [開発ロールの作成] を選択します。

15. 接続ページの「利用可能な IAM ロール」で CodeCatalyst、アカウントに追加された IAM ロールのリストに開発者ロールが表示されます。

16. [Amazon に移動] を選択します CodeCatalyst。

17. の作成ページで [CodeCatalystスペースを作成] を選択します。

スペースを編集する

スペースの説明を変更して、ユーザーがその目的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、スペース管理者権限が必要です。

このガイドの情報は、 AWS Builder ID CodeCatalyst ユーザーをサポートするスペースを編集するためのものです。ID フェデレーションをサポートするスペースをセットアップして管理する手順の詳細については、『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/ CodeCatalyst で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。スペースの説明に必要な変更を加え、[保存] を選択します。

内のスペースを削除する CodeCatalyst

スペースを削除すると、そのスペースのすべてのリソースへのアクセス権を削除できます。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**
>
> スペースの削除は元に戻せません。

スペースを削除すると、すべてのスペースメンバーは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、サードパーティのソースリポジトリから要求されたワークフローも停止されます。

> **Note**
>
> CodeCatalystスペース名は全体で一意でなければなりません。削除し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 CodeCatalyst ユーザーをサポートするスペースを削除するためのものです。ID フェデレーションをサポートするスペースをセットアップして管理する手順の詳細については、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/ CodeCatalyst で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. **`delete`**入力して削除を確定します。
5. [削除] をクリックします。

   > **Note**
   >
   > 複数のスペースに所属している場合、スペース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。

スペースのアクティビティの監視

最近作成されたプロジェクトやステータスの更新を確認するには、 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の実行や作成されたプロジェクトなどの指標を確認できます。

スペース内のアクティビティを表示するには

1. https://codecatalyst.aws/ 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. [Activity] (アクティビティ) を選択します。
4. アクティビティで情報を表示します。
5. アクティビティで絞り込むには、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。

AWS アカウント スペースの管理

Amazon AWS アカウント CodeCatalyst スペース内のリソースを使用できます。そのためには、AWS アカウント とスペース内のスペースと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スペース内のリソースと相互作用できるようになります AWS アカウント。 AWS アカウント CodeCatalyst スペースで使用したい接続を 1 つずつ作成する必要があります。

接続を作成したら、 AWS IAM ロールをその接続に関連付けることを選択できます。

トピック

- AWS アカウント スペースへの追加
- アカウント接続への IAM ロールの追加
- デプロイ環境へのアカウント接続と IAM ロールの追加
- アカウント接続を表示する
- スペース (内 CodeCatalyst) からのアカウントの削除

AWS アカウント アカウントをスペースに追加することで CodeCatalyst 、承認されたアカウントを使用するように設定できます。 AWS アカウント CodeCatalyst スペースに追加することで、 AWS アカウント プロジェクトワークフローにリソースや請求設定へのアクセスを許可できます。

AWS アカウント を追加すると、 CodeCatalyst このアカウントの使用を許可する接続が作成されます。add AWS アカウント を使用して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。
- IAM CodeCatalyst AWS ロールを引き受けてリソースにアクセスし、 AWS のサービス アカウントにデプロイできるようにします。AWS アカウント スペースの管理 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で作成されます。 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できるように接続をさらに設定します。

AWS アカウント スペースへの追加

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

AWS アカウント をスペースに追加する前に CodeCatalyst、以下の前提条件を満たしてください。

- AWS アカウント 接続するアカウントに AWS IAM ロールを作成して作成するための権限を取得します。
- アカウント接続に関連付けたい1つまたは複数のIAMロールを作成します。これには、ロールのアクセス権限を含むIAMポリシーも含まれます。
- CodeCatalyst接続を作成したいスペースのスペース管理者ロールを取得します。

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。

CodeCatalyst 接続を作成したい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必要です。追加するユーザーには管理者権限も必要です。 AWS アカウント

接続を作成する

1. で AWS Management Console、接続を確立したいアカウント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/ CodeCatalyst でコンソールを開きます。
3. CodeCatalyst 自分の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. [追加] を選択します AWS アカウント。
5. 「 AWS アカウント Amazonと関連付ける CodeCatalyst」ページの「AWS アカウント ID」に、スペースに接続したいアカウントの12桁のIDを入力します。ID の検索方法については、「 AWS アカウント 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. AWS アカウント ページが詳細ページに戻り、成功バナーが表示されます。

ステップ 2: アカウント接続リクエストを受け付ける

CodeCatalyst コンソールでへの接続要求を送信したら、 AWS 管理者と協力して AWS アカウント、提供された接続トークンを使用して接続要求を送信し、その接続要求を承認します。

アカウントの管理者権限を持っていることと、 AWS Management Console AWS アカウント 接続を作成したときと同じアカウントでにサインインしていることを確認してください。

接続リクエストを承認するには (コンソール)

1. で AWS Management Console、接続を作成したいアカウント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/ CodeCatalyst でコンソールを開きます。
3. CodeCatalyst 自分の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 [セットアップ完了] を選択します AWS Management Console。
5. に「Amazon CodeCatalyst スペースの確認」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。

   ページに直接アクセスするには、https://console.aws.amazon.com/codecatalyst/home/ AWS Management Console にある Amazon CodeCatalyst Spaces にサインインしてください。

   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) [承認済みの有料利用枠] で [有料利用枠を承認 (スタンダード、エンタープライズ)] を選択し、請求先アカウントの有料階層を有効にします。

   > **Note**
   >
   > これによって請求階層が有料階層にアップグレードされるわけではありません。ただし、これにより、 AWS アカウント スペースの請求レベルをいつでも変更できるように設定されます。 CodeCatalyst有料プランはいつでも有効にできます。この変更を行わないと、スペースは無料利用枠しか使用できません。
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認証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、追加した IAM ロールとともに接続をコンソールに表示できます。

承認された接続を確認するには

1. CodeCatalyst 自分の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、作成日とともに表示されます。
3. アカウントの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、その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:アカウント接続用に追加の IAM ロールを作成します。

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

アカウントに接続するには、作成したロールの Amazon リソースネーム (ARN) が必要です。ここで説明しているように、1 つまたは複数のロールの ARN をコピーします。IAM ロールの ARN の使用方法の詳細については、「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. 上部にあるロール ARN の値をコピーします。

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、スペース内のプロジェクトで使用したい IAM ロールの追加が含まれます。 CodeCatalyst

> **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てください。

IAM ロールをアカウント接続 (コンソール) に追加します。

1. で AWS Management Console、管理したいアカウント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/ CodeCatalyst でコンソールを開きます。
3. CodeCatalyst 自分の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、[ロールの管理元] を選択します AWS Management Console。

   「Amazon CodeCatalyst スペースに IAM ロールを追加」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権限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 CodeCatalyst で開発管理者ロールを作成」を選択します。`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。

     [開発ロールの作成] を選択します。
   - IAM ですでに作成しているロールを追加するには、[既存の IAM ロールを追加] 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/ AWS Management Console にある Amazon CodeCatalyst Spaces にサインインしてください。
7. CodeCatalyst スペースに追加し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 [使用可能な IAM ロール] に CodeCatalyst、アカウントに追加された IAM ロールのリストが表示されます。「IAM ロールを関連付ける」を選択します。 CodeCatalyst
9. 「IAM ロールを関連付ける」ポップアップの「ロール ARN」に、スペースに関連付けたい IAM ロールの Amazon リソースネーム (ARN) を入力します。 CodeCatalyst

「目的」で、アカウント接続でそのロールをどのように使用したいか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を指定しますRUNNER。SERVICE別のサービスへのアクセスに使用するロールを指定します。

目的は複数指定できます。

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

10. [IAM ロールを関連付ける] を選択します。IAM ロールを追加する場合は、これらの手順を繰り返します。

デプロイ環境へのアカウント接続と IAM ロールの追加

Amazon ECS AWS AWS Lambda やデプロイ用のリソースなどの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は、それらのリソースにアクセスする権限を持つ IAM ロールが必要です。スペース管理者ロールまたはパワーユーザーロールを使用すると、CodeCatalyst AWS アカウント リソースが作成されている場所にアカウントを接続できます。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では、IAM ロールを環境に追加する必要があります。 CodeCatalyst

デプロイ環境で使用したい IAM ロールをプロジェクトに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のデプロイ環境には追加されません。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、ステップ 4: 接続に IAM ロールを追加する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。

次に、 CodeCatalyst コンソールの Environments 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 CodeCatalyst ロールを必要とするアクションに IAM ロールを使用する場合のみ、IAM ロールを環境に追加します。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 (ビルドアクションを含む) は、環境を使用する必要があります。 CodeCatalyst

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/ CodeCatalyst でコンソールを開きます。
2. アカウント接続と IAM ロールを追加したいデプロイ環境のある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] には、環境に追加されたアカウント (存在する場合) が一覧表示されます。
6. [関連付ける] AWS アカウント を選択します。「 AWS アカウント アソシエイト先<environment_name>」ページが表示されます。
7. 「接続」で、追加する IAM ロールとの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。

アカウント接続を表示する

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/ CodeCatalyst でコンソールを開きます。
2. 表示したいアカウント接続の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 アカウントの下には、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が表示されます。

アカウント接続の詳細を表示するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. CodeCatalyst 自分の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。詳細ページには、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が表示されます。

スペース (内 CodeCatalyst) からのアカウントの削除

不要になったアカウント接続は削除できます。この手順では CodeCatalyst 、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**
>
> アカウント接続を削除すると、再接続することはできません。新しいアカウント接続を作成し、必要に応じて 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、請求先アカウントをスペース用に指定する必要があります。指定請求先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、そのスペース用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**
>
> これらの手順を使用してアカウントを削除することはできますが、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されている場合もあります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/ 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。[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リストから削除されます。

接続されたアカウントの IAM ロールの管理

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請求先アカウントを追加する場合、ロールを作成する必要はありません。

には AWS アカウント、 AWS アカウント スペースに追加したいロールを作成する権限が必要です。IAM のリファレンスやサンプルポリシーなど、IAM 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。Identity and Access Management と Amazon CodeCatalystで使用されてい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、を参照してください。 CodeCatalyst CodeCatalyst トラストモデル

では CodeCatalyst、アカウント (および該当する場合はロール) をスペースに追加する手順を完了するには、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。

以下の方法のいずれかを使用して、アカウント接続にロールを追加できます。

- CodeCatalystWorkflowDevelopmentRole-***spaceName***ロールのアクセス権限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、を参照してくださいCodeCatalystWorkflowDevelopmentRole-***spaceName*** ロール。
- ロールを作成し、ブループリント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 CodeCatalyst ロールの作成と信頼ポリシーの使用。
- IAM ロールの作成時に使用するサンプルロールポリシーのリストについては、を参照してください。AWS リソースへの Amazon CodeCatalyst アクセス用の IAM ロール
- ワークフローアクション用のロールを作成する詳細な手順については、以下の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 へのアーティファクトのアップロード
  - 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 へのアプリケーションのデプロイ
  - チュートリアル:アクションを使った lint コード GitHub

トピック

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

CodeCatalystWorkflowDevelopmentRole-*spaceName* ロール

開発者ロールは IAM の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の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。追加するユーザーには管理者権限も必要です。 AWS アカウント

以下の手順を開始する前に、 AWS Management Console CodeCatalyst スペースに追加するのと同じアカウントでにログインする必要があります。そうしないと、コンソールから不明なアカウントエラーが返されます。

を作成、追加するには CodeCatalyst CodeCatalystWorkflowDevelopmentRole-*spaceName*

1. CodeCatalyst コンソールで始める前に、を開き AWS Management Console、 AWS アカウント自分のスペースに同じものでログインしていることを確認します。
2. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
3. CodeCatalyst 自分の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 ロールを作成したい場所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 スペースに IAM ロールを追加」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。
6. [IAM CodeCatalyst で開発管理者ロールを作成] を選択します。このオプションでは、開発ロールのアクセス権限ポリシーと信頼ポリシーを含むサービスロールが作成されます。`CodeCatalystWorkflowDevelopmentRole-`*`spaceName`*ロールには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。

> **Note**
>
> このロールは開発者アカウントでのみ使用することをおすすめします。`AdministratorAccess` AWS 管理ポリシーを使用するため、 AWS アカウントこのアカウントで新しいポリシーやリソースを作成するためのフルアクセス権が付与されます。

7. [開発ロールを作成] を選択します。
8. 接続ページの「利用可能な IAM ロール」で CodeCatalyst、アカウントに追加された IAM ロールのリストにそのロールが表示されます。`CodeCatalystWorkflowDevelopmentRole-`*`spaceName`*
9. 自分のスペースに戻るには、「Go to Amazon」を選択してください CodeCatalyst。

AWSRoleForCodeCatalystSupport ロール

サポートロールは IAM の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の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。追加するユーザーには管理者権限も必要です。 AWS アカウント

以下の手順を開始する前に、 AWS Management Console CodeCatalyst スペースに追加するのと同じアカウントでにログインする必要があります。そうしないと、コンソールから不明なアカウントエラーが返されます。

を作成、追加するには CodeCatalyst AWSRoleForCodeCatalystSupport

1. CodeCatalyst コンソールで始める前に、を開き AWS Management Console、 AWS アカウント自分のスペースに同じもので ログインしていることを確認します。
2. CodeCatalyst 自分の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. AWS アカウント ロールを作成したい場所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 スペースに IAM ロールを追加」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。この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 Support ロールを追加」を選択します。このオプションでは、プレビュー開発ロールのアクセス権限ポリシーと信頼ポリシーを含むサービスロールが作成されます。AWSRoleForCodeCatalystSupport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AWSRoleForCodeCatalystSupport サービスロールについて。
6. 「 CodeCatalyst Support にロールを追加」ページで、デフォルトを選択したままにして、「ロールを作成」を選択します。

7. [利用可能な IAM ロール] で CodeCatalyst、CodeCatalystWorkflowDevelopmentRole-*spaceName*アカウントに追加された IAM ロールのリストにそのロールが表示されます。
8. 自分のスペースに戻るには、「Go to Amazon」を選択してください CodeCatalyst。

IAM CodeCatalyst ロールの作成と信頼ポリシーの使用

AWS アカウント 接続に使用する IAM ロールは、ここに記載されてい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。 CodeCatalyst 以下の手順を使用して IAM ロールを作成し、の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、ロールのアクセス権限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こともできます。CodeCatalystWorkflowDevelopmentRole-*spaceName*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. AWS Management Console にサインインし、https://console.aws.amazon.com/iam/ にある 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. [権限の追加] で、IAM ですでに作成している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。アカウント接続または環境にロールを追加するときに、この情報を提供する必要があります。

スペースユーザーの管理

スペースに参加するユーザーのロールを表示、追加、削除、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 CodeCatalyst ユーザーをサポートするスペース内のユーザーの招待と管理を目的としています。ID フェデレーションをサポートするスペースをセットアップして管理する手順の詳細については、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を表示する

表示名、エイリアス、スペースでの役割に関する情報など、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次の 3 つの役割があります。

- 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すべての権限があります。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てください。

  このロールを後で変更するには、まずユーザーを削除する必要があります。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。スペース内でプロジェクトを作成し、スペースのユーザーとリソースの管理を支援する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プロジェクトメンバーの管理。

スペース管理者テーブルには、スペース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、プロジェクトでのロールはないため、スペースメンバーには表示されません。

スペースメンバーテーブルには、スペース管理者ロールは持たないものの、プロジェクト内で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。

CodeCatalyst ユーザーは次のように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表示されます。

- スペース管理者の役割を持つユーザーが、後でプロジェクトの招待と役割を承諾しても、スペースの [スペースメンバー] テーブルや [プロジェクト] の [プロジェクトメンバー] テーブルには表示されません。そのメンバーは、いずれのスペース管理者テーブルにも引き続き表示されます。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でプロジェクトへの招待を受け入れたユーザーは、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スペース管理者ロールに変更されたが、スペースメンバーテーブルからスペース管理者テーブルにも移動する場合。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/ CodeCatalyst で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**
>
> スペース管理者権限を持っている場合は、どのプロジェクトに直接招待されたかを確認できます。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。

「ステータス」列の有効な値は次のとおりです。

- CodeCatalyst 招待済み — 招待状は送信したが、ユーザーがまだ承諾または辞退していない。
- メンバー — ユーザーが招待を承諾しました。

ユーザーをスペースに直接招待する

CodeCatalyst ユーザーをスペースに直接招待できます。これは、そのユーザーにスペース管理者またはパワーユーザーの役割を割り当てて、スペースの管理を手伝ってもらえるように招待したい場合に便利です。これらのロールのいずれかを他のユーザーに割り当てると、そのユーザーをプロジェクトに招待しなくても、スペース管理の責任をより多くの人に分散できます。

> **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。

スペース管理者テーブルには、スペース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、プロジェクトでのロールはないため、スペースメンバーテーブルには表示されません。

プロジェクトへの招待を承諾したメンバーは、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、プロジェクトで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の詳細については、を参照してくださいセットアップ CodeCatalyst。

ユーザーをスペースに招待するには

1. https://codecatalyst.aws/ CodeCatalyst で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. スペースに招待したい人のメールアドレスを入力します。「ロール」で、スペース内でそのユーザーに割り当てたいロールを選択します。
6. [招待] を選択します

スペースへの招待をキャンセルする

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

スペースへ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/ CodeCatalyst で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だけです。

5. 招待されたメンバーがいる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] を選択して確定します。

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。

スペース管理者テーブルには、スペース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/ CodeCatalyst で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. スペースメンバーテーブルで、ロールを変更したいユーザーを選択します。[ロールを変更] を選択します。

スペースメンバーを削除する

スペースのメンバーは、スペースリソースにアクセスする必要がないときに削除できます。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。

スペース管理者テーブルには、スペース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、プロジェクトでのロールはないため、スペースメンバーテーブルには表示されません。この表では、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。

Project members テーブルからユーザーを削除するには

1. https://codecatalyst.aws/ CodeCatalyst で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、そのユーザーはスペース内のすべてのプロジェクトから削除されます。また、そのプロジェクト内のリソースに関連する権限も削除されます。

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する

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。スペース管理者ロールを持つユーザーのロールを変更すると、そのユーザーは基本的にスペース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どのプロジェクトでも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**
>
> スペース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。スペース管理者ロールを持つ別のユーザーに連絡してください。

スペース管理者ロールを持つユーザーをスペース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投稿者) 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、そのユーザーはスペースから完全に削除されます。

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. 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したい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許可されていない保留中の招待がないことを確認します (ステータスが「招待済み」)。

   > **Important**
   >
   > スペース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、[削除] を選択します。

   [メンバーを削除] ダイアログボックスで、次のいずれかを実行します。

   - ユーザーのスペース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は、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メンバーになっていたすべての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。スペース管理

者ロールがユーザーの唯一のロールだった場合、そのユーザーはスペースから完全に削除されます。

チーム管理

スペースを作成したら、チームを追加できます。チームでは、ユーザーをグループ化して権限を共有したり、プロジェクト、課題管理、ロール、リソースを管理したりできます 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。

トピック

- チームを作成する
- チームを表示する
- チームのスペースロールの管理
- チームのプロジェクトロールの管理
- ユーザーをチームに直接追加する。
- ユーザーをチームから直接削除する。
- SSO グループをチームに追加します。
- チームを削除する。

チームを作成する

チームはスペース内でパワーユーザーなどのロール権限を持つことができます。チームには、プロジェクト管理者などのプロジェクト権限をプロジェクト内で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役割を持つ多数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 AWS Builder ID スペースの場合は個人ユーザー、ID フェデレーションをサポートするスペースの場合は SSO グループの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は、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は、複数のロールを持っている場合はインジケーターを表示し、最も権限の多いロールが最初に表示されます。

> **Note**
>
> スペースがアイデンティティフェデレーションをサポートしている場合は、すでに SSO ユーザーまたは SSO グループを IAM Identity Center に設定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。チームメンバーの管理には次の 2 つのオプションがあります。

- ユーザーを直接追加 — ユーザーを個別に追加または削除します。たとえば、 AWS Builder ID ユーザーまたは IAM Identity Center ですでに設定されている SSO ユーザーのいずれかを選択して、ユーザーをチームに追加します。 AWS Builder ID ユーザーまたは 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すると、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 グループの使用 — IAM Identity Center に既に設定されている 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すると、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/ CodeCatalyst で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. [チームを作成] 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なければなりません。

   (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、使用可能なスペースロールのリストから CodeCatalyst 、チームに割り当てたい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引き継がれます。

   - 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-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。

   - ユーザーを個別に管理するには、[メンバーを直接追加] を選択します。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。
   - IAM Identity Center ですでに設定している SSO グループを選択するには、「SSO グループを使用する」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。SSO グループは最大 5 つまで追加できます。

   > **Note**
   >
   > これは後で変更することはできません。 AWS Builder ID ユーザーまたは 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すると、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すると、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 グループを使用する場合、チームの作成時に SSO グループのユーザーはプルされないことに注意してください。ユーザーは、 CodeCatalyst リストに表示される前に口グインしている必要があります。

チームを表示する

では CodeCatalyst、チームのプロジェクトとロールを表示できます。メンバーページでは、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では、そのチームに関連付けられている SSO グループのリストも表示されます。

チームを表示するには

1. https://codecatalyst.aws/ CodeCatalyst で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] では、このスペースでチームに割り当てられているロールが表示されます。
4. プロジェクトロールタブでは、 CodeCatalyst チームがメンバーとして追加されたスペース内の各プロジェクトに割り当てられたプロジェクトとプロジェクトロールが表示されます (AWS ビルダー ID スペースのみ)。
5. 「メンバー」タブには、チームに割り当てられたメンバーのリストが表示されます。
6. SSO グループタブでは、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)。

チームのスペースロールの管理

チームはスペース内でパワーユーザーなどのロール権限を持つことができます。チームのスペースロールは変更できますが、チームのメンバー全員がその権限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/ CodeCatalyst で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「スペースロールを変更」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役割が変更されます。
   - 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-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。

チームのプロジェクトロールの管理

のチームは CodeCatalyst 、チームメンバーがプロジェクト管理者などのロール権限を持つことができるという点でユーザーと似ています。ロールの変更はチームに適用され、チームのすべてのメン

バーがその権限を継承します。プロジェクトごとにロールを 1 つ選択すると、自動的にチームに付与されます。

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。

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/ CodeCatalyst で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、このリストでプロジェクトの横にあるセレクターを選択し、「ロールを変更」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を追加」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 1 つを選択します。
   - 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。
   - 寄稿者-詳細については、を参照してくださいコントリビューターロール。
   - レビュー担当者-詳細については、を参照してください。レビュー担当者の役割
   - 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/ CodeCatalyst で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、そのチーム内のすべてのユーザーの関連する権限が削除されます。

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、その新しいユーザーはチーム内の既存のロールすべてから権限を継承します。

スペースが AWS Builder ID ユーザーサポート用か ID フェデレーション用かに関わらず、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が設定されている場合、「ユーザーを直接追加」を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します。

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。

ユーザーを直接追加するには:

1. https://codecatalyst.aws/ CodeCatalyst で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. [メンバーを追加] を選択します。

   > **Note**
   >
   > チームに追加するユーザーは、すでにスペースのメンバーになっている必要があります。スペースのメンバーではないチームメンバーを追加したり招待したり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。 AWS Builder ID ユーザーまたは IAM ID センターですでに設定されている SSO ユーザーを選択します。

ユーザーをチームから直接削除する。

チームメンバーをチームから削除できます。すべての権限がユーザーに継承されなくなります。ユーザーは後で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する権限は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/ CodeCatalyst で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、[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します。

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、別のチームとしてスペースに参加する SSO グループを追加できます。

> **Note**
>
> AWS Builder ID ユーザーまたは 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/ CodeCatalyst でコンソールを開きます。
2. スペースのページで [Teams] 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。SSO グループは最大 5 つまで追加できます。

チームを削除する。

不要になったチームは削除できます。

> **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の関連する権限が削除されます。

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/ CodeCatalyst で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「チームを削除」を選択します。これにより、チーム全体の役割が変わります。
4. [削除] をクリックします。

マシンリソースの管理

マシンリソースは、SSO CodeCatalyst 経由でアクセスする場合の、許可されたリソースからのユーザーの 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権限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化または無効化したりできます。たとえば、アクセスを管理するためにマシンリソースを無効化し、後で再び有効にしたい場合があります。

これらの操作は、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、マシンリソースに対して実行できます。たとえば、認証情報が漏えいした疑いがある場合は、マシンリソースを無効にできます。通常、これらの操作は使用する必要はありません。

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

トピック

- マシンリソースの表示

- マシンリソースを無効にします。
- マシンリソースを有効にします。

マシンリソースの表示

スペース内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/ CodeCatalyst で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] を選択すると、ワークフローのマシンリソースのみが表示されます。[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。

   Filter 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。

マシンリソースを無効にします。

スペース内で使用中のマシンリソースを無効化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、スペース内の関連するすべてのブループリントまたはワークフローに対するすべての権限が削除されます。

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/ CodeCatalyst で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、スペース内の関連するすべてのブループリントまたはワークフローに対するすべての権限が削除されます。

- 個別に無効化するには、無効化したい 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] 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、[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、[無効化] 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、[無効] を選択し、次に [すべてのブループリント] を選択します。

マシンリソースを有効にします。

スペース内で使用中であって無効になっているマシンリソースを有効化することを選択できます。

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/ CodeCatalyst で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 。

   - 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。[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、[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、[有効化] 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、[有効化] を選択し、次に [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 Dev Environments 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。

開発環境を管理する際の考慮事項

- [設定] の [開発環境] ページを表示したり、開発環境をスペースレベルで管理したりするには、スペース管理者ロールが必要です。
- スペースメンバーは、各自のアカウントを通じてプロジェクトで作成した開発環境を管理します。CodeCatalyst Dev Environments 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管理することになります。
- Dev Environmentsは、デフォルトで特定のコンピューティングとストレージの設定になります。構成をアップグレードする際の請求と料金については、Amazon CodeCatalyst 料金表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構成、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。の開発環境 CodeCatalyst

トピック

- スペースの開発環境を表示する
- スペースの開発環境の編集
- スペースの開発環境の停止
- スペースの開発環境を削除する

スペースの開発環境を表示する

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、開発環境をスペースレベルで管理するには、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/ 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. [設定] を選択し、次に [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。各開発環境のリソース名、リソースエイリアス (該当する場合)、IDE のタイプ、デフォルトまたは構成済みのコンピュートとストレージ、および設定済みの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Dev Environmentの実行を停止するために設定されているタイムアウトの長さ (ある場合) など、開発環境の構成を編集できます。Dev Environment の編集について詳しくは、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、開発環境をスペースレベルで管理するには、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/ 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. [設定] を選択し、次に [開発環境] を選択します。
4. 管理したい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. Dev Environment のコンピュートタイムアウト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は、実行中の開発環境をアイドル状態になる前に停止できます。そうしないと、タイムアウトが経過した開発環境はすでに停止してしまいます。開発環境の停止に関する詳細は、「」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、開発環境をスペースレベルで管理するには、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/ CodeCatalyst で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. 管理したい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を削除する

不要になった開発環境や所有者がなくなった開発環境は削除できます。Dev Environment を削除する際の考慮事項について詳しくは、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、開発環境をスペースレベルで管理するには、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/ 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. [設定] を選択し、[開発環境] を選択します。
4. 管理したい開発環境の横にあるセレクターを選択します。[削除] をクリックします。確認するには**`delete`**、入力して [Delete] を選択します。

内のスペースのクォータ CodeCatalyst

次の表では、Amazon のスペースの割り当てと制限について説明しています。 CodeCatalystAmazon 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]

ユーザー 1 人あたりのアクティブスペース数 AWS リージョン	Five
1 ユーザーあたりの 1 か月あたりのリージョンあたりのスペース作成数	Five
スペースの説明	スペースの説明は省略可能です。指定する場合は、長さが 0 ~ 200 文字でなければなりません。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、および以下の特殊文字を自由に組み合わせることができます。 `? & $ % + = / \ ; : \n \t \r`
スペース名	スペース名は全体で一意でなければなりません CodeCatalyst。削除し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 文字でなければなりません。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、ダッシュを自由に組み合わせ

	ることができます。次の文字は使用できません。 ! ? @ # $ % ^ & * ( ) + = { } [ ] | 八 > < ~ ` ' " ; :

のプロジェクト CodeCatalyst

Amazon CodeCatalyst のプロジェクトを使用して、開発チームが共有の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、リソースを追加、更新、削除できます。チームの仕事の進捗状況を監視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。

CodeCatalyst 内のスペースはプロジェクトで構成されています。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になっているプロジェクトだけです。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したユーザーに割り当てられます。

- コントリビューター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誰でも、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。
- スペース管理者、プロジェクト管理者、またはロールを持つ人なら誰でも、プロジェクトに参加するための招待状を送信できます。
- 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。
- 制限付きアクセス権限を持つユーザーは、CI/CD ワークフローの一部として、機能、コード修正、テストに関する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、ワークフローを組み立てることができます。アクションを実行すると、プロジェクトのクラウドリソースを使用して、ワークフローアクションをオンデマンドで計算できるようになります。設定したいアクティビティと出力に基づいて、さらに多くの CI/CD ワークフローを設定することもできます。たとえば、ビルドアクションとテストアクション専用のワークフローを作成して、バグを修正している間もデプロイなしでテスト結果を表示してワークフローを完了できます。次に、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する別のワークフローを作成できます。

プロジェクトを作成する場合、ブループリント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てリソースを作成することも、空のプロジェクトから開始することもできます。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、 CodeCatalyst プロジェクトリソースを追跡して使用できるように作成または設定す

るツールが決まります。プロジェクトを作成した後で、リソースを手動で追加または削除できます。以下のリソースはによって作成または構成できます CodeCatalyst。

- 課題 — プロジェクトに関連する作業は、「課題」と呼ばれる個別のレコードで追跡します。機能、タスク、バグ、その他プロジェクトのあらゆる作業について課題を作成できます。
- 通知 — 監視したいリソース、監視したいイベント、通知を受け取りたい送信先のクライアントまたはメールを選択して通知を設定します。
- 検索 — コード、課題、ユーザー、プルリクエスト、パッケージをプロジェクトで検索できます。1 つのプロジェクトを検索することも、すべてのプロジェクトを検索することもできます。
- ソースリポジトリ — プロジェクトのリポジトリにあるソースコードを操作します。ソースコードの変更をコミットしたり、指定したブランチでプルリクエストをマージしたりすると、CodeCatalyst ソースが更新されます。

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のアクティビティを追跡します。プロジェクトのアクティビティはスペースレベルで監視され、集計されます。アクティビティデータの操作の詳細については、を参照してくださいすべてのプロジェクトを表示する。

AWS プロジェクトでリソースを使用している場合は、 CodeCatalyst AWS そのアカウントを、プロジェクトのリソースを統合するための管理者権限を持つアカウントに接続できます。

プロジェクトを作成した後に、ソースリポジトリ、イシュー、その他のリソースを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。

Amazon でのプロジェクトの作成 CodeCatalyst

CodeCatalyst プロジェクトでは、共有の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したり、リソースを管理したり、問題を追跡したり、ユーザーを追加したりできます。

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

- ブループリントを使ったプロジェクトの作成
- Amazon で空のプロジェクトを作成する CodeCatalyst
- GitHubリポジトリをリンクしてプロジェクトを作成する

ブループリントを使ったプロジェクトの作成

プロジェクト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、にあるブループリントリファレンスを参照してください。プロジェクトブループリントリファレンス

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したい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. [設計図から始める] を選択します。
4. ブループリントを選択し、[次へ] を選択します。
5. [プロジェクトに名前を付ける] に、プロジェクトに割り当てる名前と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. 「プロジェクトリソース」で、共通のプロジェクトパラメータを設定します。
7. (オプション) 選択したプロジェクトパラメータに基づいて更新された定義ファイルを表示するには、「プロジェクトプレビューを生成」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。
8. (オプション) ブループリントのカードから [詳細を表示] を選択すると、ブループリントのアーキテクチャの概要、必要な接続と権限、ブループリントが作成するリソースの種類など、ブループリントに関する特定の詳細が表示されます。
9. [プロジェクトを作成] を選択します。

   プロジェクトブループリントの詳細については、を参照してください。プロジェクトブループリントリファレンス

Amazon で空のプロジェクトを作成する CodeCatalyst

リソースのない空のプロジェクトを作成し、必要なリソースを後で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。

GitHubリポジトリをリンクしてプロジェクトを作成する

CodeCatalyst GitHub ソースリポジトリにリンクする新しいプロジェクトを作成できます。その後、GitHub CodeCatalyst リンクされたソースリポジトリを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、「AWS Builder ID ユーザーをサポートするスペースの作成」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 アカウントを持っている必要があり、リンク先のリポジトリをすでに作成している必要があります。

CodeCatalyst GitHub アカウント内のソースリポジトリにリンクするプロジェクトを作成するには、次の 3 つのタスクを完了する必要があります。

1. GitHub リポジトリ拡張機能をインストールします。
2. GitHub アカウントをConnect CodeCatalyst。
3. CodeCatalyst GitHub アカウントにリンクされたプロジェクトを作成します。

GitHub リポジトリ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。

   GitHub リポジトリ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。
4. [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。

GitHub リポジトリ拡張機能をインストールしたら、 GitHub 次のステップはアカウントをスペースに接続することです。 CodeCatalyst

> **Note**
>
> GitHub 空のリポジトリやアーカイブされたリポジトリはプロジェクトでは使用できません。CodeCatalyst GitHub リポジトリエクステンションは GitHub Enterprise Server のリポジトリと互換性がありません。

アカウントをに接続するには GitHub CodeCatalyst

1. の [プロジェクトの作成] ページで CodeCatalyst、 GitHub アカウントが接続されていない場合は、プロンプトが表示されます。外部サイトに移動するには、[GitHub アカウントをConnect] を選択します GitHub。
2. GitHub GitHub 認証情報を使用して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**
   >
   > GitHub 以前にアカウントをスペースに接続したことがある場合は、再認証を求めるメッセージは表示されません。代わりに、複数のスペースのメンバーまたはコラボレーターである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、 GitHub CodeCatalyst 1つのスペースにのみ所属している場合はAmazonアプリケーションの設定ページが表示されます。 GitHub 許可したいリポジトリへのアクセスをアプリケーションに設定し、[Save] を選択します。[保存] ボタンがアクティブになっていない場合は、設定を変更してからやり直してください。

3. 現在およびfuture CodeCatalyst すべてのリポジトリへのアクセスを許可するか、 GitHub 使用したい特定の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では、future GitHub アクセスされるリポジトリを含め、 GitHub アカウントのすべてのリポジトリが含まれます。 CodeCatalyst
4. に与えられた権限を確認し 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 CodeCatalyst そのアカウントのリポジトリをプロジェクトにリンクできるようになります。

プロジェクトを作成するには

1. [プロジェクトの作成] GitHubページのアカウントドロップダウンメニューから、次のいずれかを実行します。

   - GitHub すでに接続しているアカウントを選択します。 CodeCatalyst
   - (オプション) GitHub 使用したいアカウントが表示されない場合は、[GitHub アカウントを Connect] を選択して拡張機能のページに移動します CodeCatalyst。詳細については、「GitHub でのリポジトリの使用 CodeCatalyst」を参照してください。
2. GitHub リポジトリのドロップダウンメニューでは、 GitHub GitHub 接続しているアカウントのリポジトリがドロップダウンに表示されます。 GitHub プロジェクトにリンクしたいリポジトリを選択します。
3. 「プロジェクト名」テキスト入力フィールドに、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。

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 ワークフローについては、を参照してください。でのワークフロー入門 CodeCatalyst
- Amazon S3 バケットにビルドアーティファクトをデプロイする新しいプロジェクトと同様のビルドアクションを使用するには、「」でのワークフローを使用した構築 CodeCatalyst と チュートリアル:Amazon S3 へのアーティファクトのアップロード「」を参照してください。
- 空のプロジェクトから始めて、 AWS CloudFormation 同様のサーバーレスアプリケーションをスタックデプロイでデプロイする方法については、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
- 課題計画ボードを追加するには、を参照してください。の問題点 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を表示する。
- 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のソースリポジトリ CodeCatalyst。
- ワークフローの実行の成功または失敗に関するステータスアラートを送信する通知を設定する方法については、を参照してくださいAmazon での通知の管理 CodeCatalyst。
- プロジェクトにメンバーを招待するには、を参照してくださいプロジェクトメンバーの管理。
- 開発環境を設定するには、を参照してくださいの開発環境 CodeCatalyst。

プロジェクトを表示する

CodeCatalyst スペースから、自分がプロジェクト権限を持ってい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。

まだプロジェクトを作成していない場合は、を参照してくださいAmazon でのプロジェクトの作成 CodeCatalyst。プロジェクトを作成するスペースのスペース管理者ロールが必要です。

- プロジェクト概要では、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、課題を表示できます。
- 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を表示する

自分に割り当てられた、または自分で作成した未解決の課題やプルリクエストなどのプロジェクトタスクの概要と、プロジェクトに関連する開発環境を表示するには、コンソールを使用します。

プロジェクトを表示するには、そのプロジェクトのメンバーであるか、その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、Issue、プルリクエスト、Dev Environment、Issue を表示するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. 表示したい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、自分が作成したプロジェクトタスクが表示されます。

    - [メンバー] + [すべて表示] リストを表示すると、プロジェクトメンバーのリストが表示されます。
    - 「リポジトリ」カードを表示すると、プロジェクトに関連付けられているソースリポジトリが表示されます。

- ワークフローランカードを表示すると、プロジェクトに関連するワークフローが表示されます。
- Open Pull Requests カードを表示すると、自分に割り当てられたプルリクエストや自分で作成し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も表示されます。
- 「My Dev Environments」カードを表示すると、プロジェクトに関連する開発環境の概要が表示されます。
- 課題カードを表示すると、割り当てられたタスクまたは作成したタスクの概要が表示されます。

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は、自分が権限を持っているすべてのプロジェクトを表示できます。

自分に割り当てられた、または自分で作成した未解決の課題やプルリクエストなどのプロジェクトタスクの概要と、プロジェクトに関連する開発環境を表示するには、コンソールを使用します。

プロジェクトを表示するには、そのプロジェクトのメンバーであるか、その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、Issue、プルリクエスト、Dev Environment、Issue を表示するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. 表示したい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、説明が表示されます。

プロジェクト設定を表示する

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、課題を表示できます。

自分に割り当てられた、または自分で作成した未解決の課題やプルリクエストなどのプロジェクトタスクの概要と、プロジェクトに関連する開発環境を表示するには、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、Issue、プルリクエスト、Dev Environment、Issue を表示するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. 表示したい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、説明が表示されます。

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、アクセス権のあ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ーを選択します。
2. ドロップダウンを展開して、移動先のプロジェクトを選択します。

Amazon でプロジェクトを削除する CodeCatalyst

プロジェクトを削除して、その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の役割が必要です。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。

プロジェクトを削除するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. 表示したい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**と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。

プロジェクトメンバーの管理

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。ユーザーの追加または削除、現在のメンバーの役割の管理、プロジェクトへの招待の送信、まだ承認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は、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は、複数のロールを持っている場合はインジケーターを表示し、最も権限の多いロールが最初に表示されます。

プロジェクト内のメンバーを表示する

プロジェクトにユーザーを追加するときは、以下のようにプロジェクト権限を付与するロールを割り当てます。

- プロジェクト管理者ロールには、プロジェクトのすべての権限があります。このロールは、プロジェクト設定の編集、プロジェクト権限の管理、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てください。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。
- 寄稿者ロールには、プロジェクトでの作業に必要な権限があります。このロールは、プロジェクト内のコード、ワークフロー、課題、アクションを扱う必要があるユーザーに割り当ててください。詳細については、「コントリビューターロール」を参照してください。
- レビュー担当者ロールにはレビュー権限があります。詳細については、「レビュー担当者の役割」を参照してください。
- 読み取り専用ロールには読み取り権限があります。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。

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。

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/ CodeCatalyst でコンソールを開きます。

2. 表示したい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、プロジェクトで役割を持つすべてのメンバーが表示されます。

   > **Tip**
   >
   > スペース管理者権限を持っている場合は、どのプロジェクトに直接招待されたかを確認できます。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。

   スペース管理者テーブルには、スペース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自動的に (暗黙的に) スペース内のすべてのプロジェクトに割り当てられ、プロジェクトでの役割はありません。

   Status 列には、以下の値が有効です。

   - CodeCatalyst 招待済み — 招待状は送信したが、ユーザーがまだ承諾または辞退していない。
   - メンバー — ユーザーが招待を承諾しました。

トピック

ユーザ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、「Amazon のロールを使った作業 CodeCatalyst」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、「Amazon のロールを使った作業 CodeCatalyst」をご参照ください。

招待をキャンセルする

最近招待状を送信した場合は、招待状がまだ受理されていない限りキャンセルできます。

プロジェクトへ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の役割が必要です。

プロジェクトメンバーの招待をキャンセルするには

1. 招待状を送ったもののキャンセル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」タブを表示し、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だけです。

4. 招待されたメンバーがいる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] を選択して確定します。

プロジェクトからユーザーを削除する。

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、そのユーザーは自動的にそのスペースから削除されます。

プロジェクトからユーザーを削除するには:

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. 表示したい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、「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を承諾または辞退する。

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. メール内のプロジェクトリンクを選択します。
3. [承認] または [拒否] を選択します。

   [拒否] を選択すると、招待を辞退したことを知らせ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。

プロジェクトのチーム管理

プロジェクトを作成したら、チームを追加できます。チームでは、ユーザーをグループ化して権限を共有したり、プロジェクト、課題管理、ロール、 CodeCatalyst リソースをプロジェクトメンバーやスペースメンバーとして管理したりできます。

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の役割が必要です。

トピック

- プロジェクトにチームを追加する
- チームのプロジェクトロールを管理します。
- チームのプロジェクトロールを削除する

プロジェクトにチームを追加する

チームメンバーがプロジェクト内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は、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は、複数のロールを持っている場合はインジケーターを表示し、最も権限の多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/ 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。[プロジェクト設定] を選択し、[チーム] を選択します。
3. [チームを追加] を選択します。
4. 「チーム」で、参加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、で利用できる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。
   - 寄稿者 — 詳細については、を参照してくださいコントリビューターロール。
   - レビュー担当者 — 詳細については、を参照してください。レビュー担当者の役割
   - 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. [チームを追加] を選択します。

チームのプロジェクトロールを管理します。

チームはスペース内でパワーユーザーなどのロール権限を持つことができます。チームのスペースロールは変更できますが、チームのメンバー全員がその権限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/ CodeCatalyst で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、リスト内のチームの横にあるセレクターを選択し、「ロールを変更」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を追加」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 1 つを選択します。
   - 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。
   - 寄稿者-詳細については、を参照してくださいコントリビューターロール。
   - レビュー担当者-詳細については、を参照してください。レビュー担当者の役割

- 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を削除する

では CodeCatalyst、チームのプロジェクトロールを表示できます。チーム内のメンバーも表示できます。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/ CodeCatalyst で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、そのチーム内のすべてのユーザーの関連する権限が削除されます。

5. [保存] を選択します。

マシンリソースの管理

マシンリソースは、SSO CodeCatalyst 経由でアクセスする場合の、許可されたリソースからのユーザーの 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、プロジェクト内のリソースに権限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化または無効化したりできます。たとえば、アクセスを管理するためにマシンリソースを無効化し、後で再び有効にしたい場合があります。

これらの操作は、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、マシンリソースに対して実行できます。たとえば、認証情報が漏えいした疑いがある場合は、マシンリソースを無効にできます。通常、これらの操作は使用する必要はありません。

このページを表示したり、プロジェクトレベルでマシンリソースを管理したりするには、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。

トピック

マシンリソースの表示

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。

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。

マシンリソースを表示するには

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、[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] を選択すると、ワークフローのマシンリソースのみが表示されます。[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。

   Filter 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。

マシンリソースを無効にします。

プロジェクトで使用中のマシンリソースを無効にすることができます。

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

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。

マシンリソースを無効にするには

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、[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、スペース内の関連するすべてのブループリントまたはワークフローに対するすべての権限が削除されます。

- 個別に無効化するには、無効化したい 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] 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、[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、[無効化] 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、[無効] を選択し、次に [すべてのブループリント] を選択します。

マシンリソースを有効にします。

プロジェクトで使用中かつ無効になっているマシンリソースを有効化できます。

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。

マシンリソースを有効にするには

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、[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 。

- 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。[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、[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、[有効化] 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、[有効化] を選択し、次に [すべてのブループリント] を選択します。

のプロジェクトのクォータ CodeCatalyst

次の表は、Amazon のプロジェクトのクォータと制限を示しています。 CodeCatalystAmazon 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

スペースあたりの最大プロジェクト数	100
1 人のユーザーが所属できるプロジェクトの最大数	1,000
1 つのプロジェクトに所属でき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なければなりません。名前は 3 ~ 63 文字でなければなりません。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まる必要があります。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) プロジェクト名には以下の文字は使用できません。 ! ? @ # $ % ^ & * ( ) + = { } [ ] | \ / > < ~ ` ' " ; :
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。

での通知の操作 CodeCatalyst

でプロジェクトとリソースを監視する通知を設定できます CodeCatalyst。ユーザーは、自分がメンバーになっているどのプロジェクトでも、メールを受信したいプロジェクトイベントを選択できます。 CodeCatalyst スペースと Slack ワークスペース間のアクセスを設定し、プロジェクトの通知をその Slack ワークスペース内の 1 つ以上のチャンネルに送信するように設定することで、Slack などのチームメッセージングアプリケーションでチーム全体に送信され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

バーは自分の Slack メンバー ID を追加して、接続された Slack CodeCatalyst ワークスペースやチャンネルでのイベントについて直接通知を受けることができるようになります。

> **Note**
>
> Slack 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

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セットアップ CodeCatalyst。
- CodeCatalyst プロジェクト。詳細については、「Amazon でのプロジェクトの作成 CodeCatalyst」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、「Amazon のロールを使った作業 CodeCatalyst」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
- チャンネルに送信される通知の管理

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。

内のブループリント CodeCatalyst

ブループリントは、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です。 CodeCatalyst 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、サンプルコードを含むプロジェクト全体まで、あらゆるもので構成できます。ブループリントは任意のオプションセットを受け取り、それらを使用して任意の出力コードセットを生成し、プロジェクトに転送します。ブループリントが最新のベストプラクティスや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の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、ビルドおよびテストレポート、統合問題追跡ツールを含む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、設定パラメータセットに基づいてリソースとソースコードを生成します。 CodeCatalyst管理された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のほか、 CodeCatalyst 作成または設定を行うツールが決まるため、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したり、既存のプロジェクトに適用したりできます。CodeCatalyst プロジェクトには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。たとえば、Web 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を作成し、後でセキュリティブループリントを適用することができます。ブループリントの 1 つが更新されると、ライフサイクル管理を通じて変更や修正を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、「プロジェクトブループリントリファレンス」および「ブループリントユーザーとしてライフサイクル管理を操作する」を参照してください。

ブループリントの作成者は、 CodeCatalyst スペースメンバーがプロジェクトリソースを使用できるようにカスタムブループリント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特定の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は、ブループリントを管理して更新を続けることができるため、スペースのプロジェクトが最新のベストプラクティスに従って常に最新の状態に保たれます。詳細については、「でのカスタムブループリントの操作 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確認するには、オープンソースリポジトリを参照してください。 GitHub

トピック

- ブループリントを使ったプロジェクトの作成
- プロジェクトへのブループリントの適用と関連付けの解除
- プロジェクト内のブループリントの更新

- プロジェクト内のブループリントの説明の編集
- ブループリントユーザーとしてライフサイクル管理を操作する
- プロジェクトブループリントリファレンス
- でのカスタムブループリントの操作 CodeCatalyst
- でのブループリントのクォータ CodeCatalyst

ブループリントを使ったプロジェクトの作成

Amazon CodeCatalyst カタログまたはチームのスペースカタログにある設計図とカスタム設計図を使用して、プロジェクトをすばやく作成できます。ブループリントによっては、特定のリソースを使用してプロジェクトが作成されます。詳細については、「ブループリントを使ったプロジェクトの作成」および「プロジェクトブループリントリファレンス」を参照してください。

プロジェクトを作成したら、 CodeCatalyst CodeCatalyst カタログまたはカスタムブループリントを使用してスペースのカタログからプロジェクトに追加のブループリントを適用できます。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取り入れることができます。また、これによって、進化するコンポーネントに加えられた最新の変更がプロジェクトに適用されていることを確認できます。プロジェクトでのブループリントの操作について詳しくは、を参照してください。ブループリントユーザーとしてライフサイクル管理を操作する

プロジェクトへのブループリントの適用と関連付けの解除

1 つの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てソフトウェアコンポーネントを機能させる必要が減ります。また、要件が変化してもプロジェクトを最新の状態に保つことができます。ブループリントからのリソースが不要になったら、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。ブループリントをプロジェクトに適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ライフサイクル管理を操作する

トピック

- ブループリントをプロジェクトに適用する
- ブループリントとプロジェクトの関連付けを解除する

ブループリントをプロジェクトに適用する

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/ CodeCatalyst で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、ブループリント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 [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。
4. [ブループリント] CodeCatalyst タブからブループリントを選択するか、[Space ブループリント] タブからカスタムブループリントを選択し、[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。最新バージョンが自動的に選択されます。
6. [ブループリントの設定] で、ブループリントパラメータを設定します。
7. 現在のブループリントバージョンと更新したバージョンとの違いを確認します。プルリクエストに表示される差異は、現在のバージョンと最新バージョン (プルリクエストが作成された時点で必要なバージョン) との変更を示しています。変更が表示されない場合は、バージョンが同じか、現在のバージョンと目的のバージョンの両方で同じバージョンを選択した可能性があります。
8. 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、[Apply blueprint] 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。ファイルなどのリソースへのリンクを追加するには、@記号に続けてファイル名を指定します。

> **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、「プルリクエストのレビュー」および「プルリクエストをマージする」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の作成や既存のプロジェクトへの適用に使用できるブループリントがない指定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。

ブループリントとプロジェクトの関連付けを解除する

ブループリントを新たに更新したくない場合は、ブループリントとプロジェクトとの関連付けを解除できます。ブループリントからプロジェクトに追加されたリソースと機能的なソフトウェアコンポーネントは、プロジェクト内でメインになります。

ブループリントとプロジェクトとの関連付けを解除するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. CodeCatalyst コンソールでスペースに移動し、ブループリント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. 関連付けを解除したい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て、[ブループリントの関連付け解除] を選択します。
5. と入力して関連付け解除を確定しますconfirm。
6. [確認] を選択します。

プロジェクト内のブループリントの更新

ブループリントを使用してプロジェクトを作成したり、ブループリントを既存のプロジェクトに適用したりすると、ブループリント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確認できます。ライフサイクル管理では、プロジェクトに適用されている 1 つ以上のブループリントを更新できるため、プロジェクトの他の部分に影響を与えずに各ブループリントを更新できます。ブループリントの更新は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ライフサイクル管理を操作する」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/ CodeCatalyst でコンソールを開きます。
2. CodeCatalyst コンソールで、ブループリントのバージョンを更新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、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. [アクション] ドロップダウンメニューを選択し、[バージョンを更新] 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、更新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. [ブループリントの設定] で、ブループリントパラメータを設定します。
8. 現在のブループリントバージョンと更新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 (プルリクエストの作成時に必要なバージョン) との間の変更点です。変更が表示されない場合は、バージョンが同じか、現在のバージョンと目的のバージョンの両方で同じバージョンを選択した可能性があります。
9. 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、[Apply update] 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に追加することも、ファイル内の個々の行やプルリクエスト全体に追加することもできます。ファイルなどのリソースへのリンクを追加するには、@記号に続けてファイル名を指定します。

> **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更新されません。詳細については、「プルリクエストのレビュー」および「プルリクエストをマージする」を参照してください。

> **Note**
>
> ブループリントバージョンを更新するために既存のプルリクエストを開いている場合は、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てください。Update version 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に移動します。保留中のプルリクエストは、プロジェクト設定の [ブループリント] タブと [プロジェクトサマリー] ページから確認することもできます。詳細については、「プルリクエストを表示する」を参照してください。

プロジェクト内のブループリントの説明の編集

プロジェクトの作成に使用したブループリントの説明や、プロジェクトの作成後に適用したブループリントの説明を編集できます。ブループリントはプロジェクト内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、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。

プロジェクト内のブループリントの説明を編集するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. CodeCatalyst コンソールでスペースに移動し、更新するブループリント設定を含む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. 更新する説明のある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. ブループリントの説明テキスト入力フィールドに、プロジェクト内のブループリント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ライフサイクル管理を操作する

ライフサイクル管理とは、ブループリントの更新されたオプションや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、ブループリントの作成者は、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。たとえば、セキュリティ修正を Web アプリケーションブループリントにプッシュすると、その Web アプリケーションブループリントを含む、または Web 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が、その修正を自動的に適用できるようになります。これと同じ管理フレームワークにより、ブループリントユーザーはブループリントのオプションを選択した後で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ライフサイクル管理を使用する
- プロジェクト内の複数のブループリントでライフサイクル管理を使用する
- ライフサイクルのプルリクエストにおけるコンフリクトへの対処
- ライフサイクル管理の変更をオプトアウトする
- プロジェクト内のブループリントのライフサイクル管理をオーバーライドする

既存のプロジェクトでライフサイクル管理を使用する

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。たとえば、ブループリントから作成されたことのない 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ブループリ

ントを追加できます。このブループリントは、セキュリティスキャンのワークフローとその他の関連コードを生成します。これで、ブループリントに変更が加えられるたびに、Java アプリケーションのコードベースのその部分がチームのベストプラクティスに従って自動的に最新の状態に保たれます。

プロジェクト内の複数のブループリントでライフサイクル管理を使用する

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、同じプロジェクトで複数のブループリントを一緒に使用できることがよくあります。たとえば、ある会社のプラットフォームエンジニアが構築した一元的な Web API ブループリントと、アプリ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をプロジェクトで構成できます。これらのブループリント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を記憶します。

> **Note**
>
> ブループリントは任意のアーキテクチャコンポーネントであるため、すべてのブループリントが相互にマージを試みても、相互に意味があるわけではなく、論理的に連携するわけでもありません。

ライフサイクルのプルリクエストにおけるコンフリクトへの対処

ライフサイクルのプルリクエストでマージコンフリクト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。解決内容は、その後のブループリントの更新時に記憶されます。

ライフサイクル管理の変更をオプトアウトする

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参照をすべて解除し、ライフサイクルの更新をオプトアウトできます。安全上の理由から、ブループリントから追加されたものも含め、プロジェクトのコードやリソースが削除されたり、影響が及ぶことはありません。詳細については、「ブループリントとプロジェクトの関連付けを解除する」を参照してください。

プロジェクト内のブループリントのライフサイクル管理をオーバーライドする

プロジェクト内の特定のファイルに対するブループリントの更新を上書きしたい場合は、所有権ファイルをリポジトリに追加できます。GitLabのコードオーナー仕様が推奨ガイドラインです。ブループリントでは常にコード所有者ファイルを優先し、次のようなサンプルファイルを生成できます。

```
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、.ownership-file以下の内容のが生成されます。

```
[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/**
```

プロジェクトブループリントリファレンス

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、サンプルソースコード、CI/CD ワークフロー、ビルドおよびテストレポート、 CodeCatalyst 統合問題追跡ツールを含む完全なプロジェクトが作成されます。プロジェクトブループリントは、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、「Amazon で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

- 利用可能なブループリント
- プロジェクトブループリント情報の検索

利用可能なブループリント

ブループリント名	ブループリントの説明
ASP.NET コアウェブ 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。ブループリントでは.NET AWS 用のデプロイツールを使用し、Amazon Elastic Container Service AWS App Runner、 AWS Elastic Beanstalk またはをデプロイターゲットとして設定するオプションを提供しています。
AWS Glue ETL	この設計図では、 AWS CDK、 AWS Glue、AWS Lambda、Amazon Athena を使用して、カンマ区切り値 (CSV) を Apache Parquet に変換するサンプル抽出トランスフォームロード (ETL) リファレンス実装を作成しています。
DevOps デプロイパイプライン	このブループリントは、 AWS デプロイパイプライン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。これにより、 AWS リファレンスアプリケーションを複数のステージにデプロイできます。
Java API と AWS Fargate	このブループリントは、コンテナ化された Web 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 DynamoDB を基盤とするコンテナ化された Spring Boot Java ウェブサービスを構築し、Amazon ECS にデプロイしています。このプロジェクトは、コンテナ化されたアプリケーションをサーバーレスコンピューティング上の Amazon ECS クラスターにデプロイします。 AWS Fargate アプリは 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、サンプル Web サービ

ブループリント名	ブループリントの説明
	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	この設計図は、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 Python で生成し、適切に設計された 3 層のモダン Web 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	この設計図は、.NET CLI Lambda AWS Lambda ツールを使用して関数を作成します。設計図には、C# または F# の選択肢など、AWS Lambda 関数のオプションが用意されています。
Node.js API と AWS Fargate	このブループリントはコンテナ化された Web 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 ウェブサービスを構築し、Amazon エラスティックコンテナサービスにデプロイします。このプロジェクトは、コンテナ化されたアプリケーションをサーバーレスコンピューティング上の Amazon ECS クラスターにデプロイします。AWS Fargate ワークフローが正常に実行されると、サンプル Web 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)	この設計図では、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。プログラミング言語として、SDK for Java K TypeScript、または Python の場合は SDK を選択できます。

ブループリント名	ブループリントの説明
サーバーレスイメージハンドラー	この設計図は、画質を低下させずに高速画像処理を行う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	この設計図は、To Do サービスリファレンスを使用する AWS Lambda REST API を作成します。 Amazon API Gateway プログラミング言語として、SDK for Java K TypeScript、または Python の場合は SDK を選択できます。
単一ページのアプリケーション	この設計図は、React、Vue、および Angular フレームワークを使用するシングルページアプリケーション (SPA) を作成します。ホスティングには、「 AWS Amplify ホスティング」または Amazon CloudFront 「Amazon S3」を選択します。
静的ウェブサイト	この設計図は、Hugo または Jekyll の静的サイトジェネレータを使用して静的 Web 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 Web ページを生成します。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量の多いコンテンツに最適です。AWS CDK ブループリントはを使用して、静的ウェブページを Amazon S3 + AWS Amplify CloudFront のいずれかにデプロイします。
ウェブアプリケーションを実行するには	この設計図は、フロントエンドとバックエンドのコンポーネントを含む To Do サーバーレス Web アプリケーションを作成します。プログラミング言語として、SDK for Java K TypeScript、または Python の場合は SDK を選択できます。

ブループリント名	ブループリントの説明
V ideo-on-demand ウェブサービス	この設計図は、コンテンツを取り込み、トランスコードし、 video-on-demand 配信する機能を提供するサービスを作成します。ブループリントでは AWS Lambda、Amazon S3、 Amazon CloudWatch、 AWS Elemental MediaConvertを使用しています。
外部ブループリントを購読します。	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 1 回実行され、NPM にパッケージの新しいバージョンがないか確認します。新しいバージョンが存在する場合、CodeCatalyst ワークフローはそのバージョンをカスタムブループリントとして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、またはブループリントではない場合、アクションは失敗します。ターゲットパッケージは NPM 上にある必要があり、パッケージはブループリント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購読する必要があります。
Bedrock GenAI チャットボット	この設計図は、Amazon BedrockとAnthropicのClaudeによるジェネレーティブAIチャットボットを作成したものです。この設計図により、データに合わせてカスタマイズできる、安全でログイン保護された独自の LLM Playground 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。

ブループリント名	ブループリントの説明
AWS プロジェクト開発キット (AWS PDK) ブループリント	これらの PDK ブループリントを組み合わせて、React ウェブサイト、Smithy API、および 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できます。AWS PDK には、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が用意されています。詳細については、AWS PDK GitHub チュートリアル:コンポーザブル 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 ソースリポジトリとを参照してください。

プロジェクトブループリント情報の検索

には、複数のプロジェクトブループリントがあります。 CodeCatalyst各ブループリントには、サマリーと README ファイルが添付されています。概要に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が記述され、README ファイルにはブループリントの詳細と使用方法が記載されています。

でのカスタムブループリントの操作 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は、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するために使用できます。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したり、既存のプロジェクトに適用したりすると、ブループリントに加えた変更はすべてプルリクエストの更新としてそれらの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確認できるため、標準がプロジェクト間で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、各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や修正を含むベストプラクティスに従い続けることができます。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。

カスタムブループリントを使用すると、再統合によってブループリントのバージョンを前のプロジェクトと照合して更新できます。再統合とは、ブループリント合成を更新したバージョンで再実行するプロセスや、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確認するには、オープンソースリポジトリを参照してください。 GitHub

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

- 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、「プロジェクトブループリントリファレンス」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

設計図を作成するプロセス中に、設計図を設定し、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、デフォルトで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含まれています。

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 Enterprise 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、Space 管理者または Power ユーザーロールが必要です。詳細については、「Amazon のロールを使った作業 CodeCatalyst」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

- a. https://codecatalyst.aws/ で CodeCatalyst コンソールを開きます。
  b. 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。
  c. 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。
  d. ブループリントの作成を選択します。
  e. ブループリントの名前に、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。
  f. ブループリントの詳細 で、次の操作を行います。
    i. 「ブループリントの表示名」テキスト入力フィールドに、スペースのブループリントカタログに表示される名前を入力します。
    ii. 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。
    iii. 作成者名のテキスト入力フィールドに、カスタムブループリントの作成者名を入力します。
    iv. (オプション) 詳細設定 を選択します。
      A. + 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。
      B. 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。
      C. 「設計図のパッチ名」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。
      D. デフォルトでは、リリースワークフローは、プロジェクト内でブループリントビルダーと呼ばれる公開ブループリントを使用して生成されます。公開アクセス許可はリリースワークフローで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。ワークフローの

生成をオフにするには、リリースワークフローのチェックボックスをオフにします。

g. (オプション) 設計図プロジェクトには、スペースカタログへの設計図の公開をサポートする事前定義されたコードが付属しています。選択したプロジェクトパラメータに基づいて更新を含む定義ファイルを表示するには、「ブループリントプレビューの生成」から「コードの表示」または「ワークフローの表示」を選択します。

h. 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、ワークフローはブループリントの作成時に自動的に実行が開始されます。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るようになります。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、「プロジェクトへのブループリントの適用と関連付けの解除」を参照してください。

と呼ばれる公開ワークフロー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、「プロジェクトへのブループリントの適用と関連付けの解除」および「ワークフローでの作業」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を作成するときに生成され、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付属する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設計図と、それらが生成するコードは、お客様の責任となります。

ブループリント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. 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、「開発環境用に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、作成した特徴量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中と表示され、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、「の開発環境 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. 動作中のターミナルで、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している場合は、まず Yarn をシステムに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、設定に合わせて構成します。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、「カスタムブループリントの開発」、「ウィザードの使用」、および「カスタムブループリントの公開」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セットアップして開発したら、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動作であることを確認できます。

カスタムブループリントをプレビューするには

1. 動作中のターミナルで、次のyarnコマンドを使用します。

```
yarn blueprint:preview
```

2. 表示された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 が期待どおりに表示されてい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、それをスペースに公開できます。これにより、カタログに非プレビューバージョンを追加する前に、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わずに公開できます。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 として公開して追加できます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

表示された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、公開したら、最終的なブループリントバージョンを公開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。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追加と削除」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。セットアップ 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、AWS Builder ID ユーザーをサポートするスペースの作成、スペースユーザーの管理、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、「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、「プルリクエストのレビュー」および「プルリクエストをマージする」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ライフサイクル管理を使用する「」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ライフサイクル管理を使用する。

トピック

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Yan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、再合成「」および「」を参照してください再合成によるファイルの生成。

マージ戦略の使用

トピック

再合成によるファイルの生成

再合成では、設計図によって生成されたソースコードを、設計図と同じ によって以前に生成されたソースコードとマージして、設計図への変更を既存のプロジェクトに反映できます。マージは、設計図の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 つのバンドルを生成します。yarn blueprint:resynth コマンドを使用して手動でローカルで実行でき、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、再合成動作をローカルでモックしてテストできます。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為的に構築され、デプロイ先のプロジェクト内の内容 (ある場合) に関するブループリント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は、リセットされ、モックと見なされます。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と合成された場合に設計図の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、祖先は存在しないため、 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すでに存在する場合、モックとして見なされます。
- 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、ブループリントを模倣するバンドルです。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で作成され、 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。

双方向マージは、 ancestor-bundle proposed-bundleと との違いを考慮し、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、 中の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によって提供され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ファイル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の検出を試みます。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の 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

再合成中、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にマージ戦略を指定することで、所有しているファイルと所有してい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、再合成中に最新の戦略を使用できます。

```
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。未解決のファイルは、デフォルトで Git に似た three-way-merge になります。MergeStrategies コンストラクトにはいくつかのマージ戦略が用意されていますが、独自のマージ戦略を記述することもできます。提供されている戦略は、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いずれか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、標準の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、それらを単一の解決済みファイルにマージする戦略関数を作成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/**',
            ],
          },
        ],
```

コンテキストオブジェクトの使用

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。ブループリント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));
    }
  }
```

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付属する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. 設計図の作成時にリリースワークフローをオプトインすると、最新の設計図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) を生成します。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、「ウィザードの使用」、「環境コンポーネント」、「リージョンコンポーネント」、「リポジトリとソースコードのコンポーネント」、「ワークフローコンポーネント」、「開発環境コンポーネント」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ときに、追加のサポートを受けるために、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 GitHub リポジトリ」を参照してください。

ウィザードの使用

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、 Options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。ウィザードは、可能な限り最適と記述された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
- サポートされる TypeScript 型
- 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Optionsのカスタムブループリントの でサポートされています。

@inlinePolicy ./path/to/policy/file.json

- 必須 - オプションは 型である必要がありますRole。
- Usage - 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であると予想されます。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 - オプションは 型である必要がありますRole。
- Usage - 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であると予想されます。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

- 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またはその他の工ラーが必要です。
- Usage - @validation* 失敗時に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 boolean (オプション)

- 必須 - 該当なし
- Usage - サブオプションを折りたたむことができるブール値。折りたたまれた注釈が存在する場合、デフォルト値は true です。値を に設定すると、最初に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 - 変更オプションの表示名。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 - 変更オプションの表示名。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 - 指定し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 番号 (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、テキストのセクションを拡大します。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
- Usage - 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 - 注釈はブール値プロパティ上にある必要があります。選択時に true として合成す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 - アカウント接続タイプでのみ使用できます。非表示の名前の入力を表示します。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 - 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
- Usage - 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
- Usage - 現在のブループリントの要件として指定された blueprintNameをプロジェクトに適用するよう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 {
...
```

サポートされる TypeScript 型

フロントエンドウィザードのカスタムブループリント では`Options`、次の TypeScript タイプがサポートされています。

数

- 必須 - オプションは 型である必要があります`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 - オプションは 型である必要があります`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 - オプションは 型である必要がありますboolean。
- Usage - チェックボックスを生成します。
- 例 - `isProduction: boolean`

```
{
  isProduction: boolean
  ...
}
```

ボストン

- 必須 - オプションは 3 つ以下の文字列の和集合である必要があります。
- Usage - 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ある必要があります。
- Usage - 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する必要があります。
- Usage - 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 - オプションは 型である必要があります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 - オプションは 型の配列である必要があります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 - オプションは 型である必要があります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 - オプションは 型である必要がありますSelector。
- Usage - 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、ユーザーと通信し直すことができます。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ます。基本ブループリントには、明確なエラーメッセージを想定したthrowSynthesisError(...)関数が実装されています。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.'
})
```

環境コンポーネント

カスタムブループリントウィザードは、ウィザードで公開された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 ファイルに以下を追加します。

```
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 を使用して生成されます。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

ザーがアカウントを接続していない場合や、アカウントをロールに接続していない場合は確認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、ブループリントが完全に展開されます。テストおよび開発の目的で、フルモックを defaults.json ファイルに含めると便利です。

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'];
```

```
    };
  };
}
```

モックインターフェイスを完了する

```
{
  ...
  "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"
        ]
      }
    }
  },
}
```

シークレットコンポーネント

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て、ワークフローで参照できます。詳細については、「シークレットの使用」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` ファイルに以下を追加します。

```
import { Secret, SecretDefinition } from '@amazon-codecatalyst/blueprint-
component.secrets'
```

トピック

- シークレットの作成
- ワークフローでのシークレットの参照

シークレットの作成

次の例では、ユーザーにシークレット値とオプションの説明の入力を求める 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"
    },

}
```

ワークフローでのシークレットの参照

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、「ワークフロー内のシークレットの参照」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シークレットの使用。

リージョンコンポーネント

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 ファイルに以下を追加します。

```
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

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、select/multiselect コンポーネントのプレースホルダーを変更できます。

  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のリージョンの選択

```
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']>[];
}
```

リポジトリとソースコードのコンポーネント

リポジトリは、コードを保存 CodeCatalyst するために Amazon によって使用されます。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) などのその他のコンポーネントなど、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、「のソースリポジトリ 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` ファイルに以下を追加します。

```
import {...} from '@caws-blueprint-component/caws-source-repositories'
```

トピック

ファイルの追加

テキストファイルは、 SourceFileコンストラクトを使用して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可能性のある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

リポジトリには任意のビットを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

スターターコードをコピーして貼り付け、そのベースの上にさらにコードを生成することで、生成されたコードの使用を開始できます。コードを static-assets ディレクトリに配置し、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StaticAssetは ですSubstitutionAsset。サブクラスはまったく同じように機能しますが、代わりに ファイルに対して mustache 置換を実行できます。 copy-and-replace スタイル生成を実行するのに便利です。

静的アセット置換では、生成された出典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、タッ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、トリプルの mustache 構文 を使用します{{{name}}}。詳細については、「 mustache templating rules」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 というサブクラスによる glob ターゲット設定をサポートします。このサブクラスはfindAll(...)、パス、コンテンツなどとともにプリロードされた静的アセットのリストを返します。リストをFile構造でチェーンして、 static-assets ディレクトリにコンテンツを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と、 という名前myRepoの単一のリポジトリが生成され、ルートにソースファイルREADME.mdと CodeCatalystワークフローが生成されます。

ワークフローコンポーネント

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、「ワークフロー定義リファレンス」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 ファイルに以下を追加します。

```
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作成できます。定義は、リポジトリでレンダリングするワークフローコンポーネントに指定できます。

```
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で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

マネージド開発環境 (MDE) は、 で MDE Workspace を作成および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、「Devfile 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開発ファイルの移動。

```
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 ファイルに以下を追加します。

```
import {...} from '@amazon-codecatalyst/codecatalyst-workspaces'
```

コンポーネントの問題

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、検索、グループ化、フィルタリングが可能な説明、担当者、ステータス、およびその他のプロパティを含めることができます。問題は、デフォルトのビューを使用して表示することも、カスタムフィルタリング、ソート、または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、課題の概念「」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 フィールドと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 ファイルに以下を追加します。

```
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

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、ブループリントの作成者として提供される設定のスナップショットテストをサポートします。設定は部分的な上書きであり、設計図のルートにある defaults.json ファイルの上にマージされます。スナップショットテストが有効で設定されている場合、ビルドおよびテストプロセスは指定された設定を合成し、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「ブループリント 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. ファイルで.projenrc.ts、スナップショットするファイル 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。Projen によって維持および再生成された出典ファイルを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含むdefault-config.jsonファイルを表示します。ファイルを更新するか、1 つ以上の独自のテスト設定に置き換えます。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きは、次の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、次のメッセージが表示されます: Snapshots: NN passed, NN total。

意図的に設計図を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で一定であることを前提としています。設計図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**
>
> globs 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、デフォルトのエントリを明示的に元に戻す必要がある場合があります。

カスタムブループリントの公開

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、プロジェクトのソースコード、設定、およびリソースを表し、 CodeCatalyst デプロイ API オペレーションが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。

> **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付属する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設計図と、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

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、それをスペースに公開できます。これにより、カタログに非プレビューバージョンを追加する前に、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

ンを構成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 として公開して追加できます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. (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、ブループリント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
   ```

   表示された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 が期待どおりに表示されてい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、それを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しなくても、プロジェクト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
```

設計図は 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
```

設計図は 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、「カスタムブループリントをスペースカタログに追加する。」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た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。設計図の作成時にワークフローの生成をオフにした場合、設計図が自動的にスペースの設計図カタログに追加されることはありません。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公開されます。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後、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、「カスタムブループリントをスペースカタログに追加する。」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、 blueprint:previewおよび 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 Enterprise スペースがある場合は、それらのスペースに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。また、ブループリントを別のスペースの既存のプロジェクトに適用することもできます。

指定され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のバージョンを発行します。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

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。

スペースへのカスタムブループリントの追加と削除

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。カスタムブループリントを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する必要がなくなった場合は、スペースのブループリントカタログから削除することもできます。

トピック

- カスタムブループリントをスペースカタログに追加する。
- 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をスペースカタログに追加する。

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、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ときに、そのブループリントをすべてのスペースメンバーが使用できるようになります。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する前に、ブループリントの公開権限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選択した場合、公開権限はデフォルトで有効になっています。詳細については、「カスタムブループリントの公開権限の管理」および「カスタムブループリントの公開」を参照してください。

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には

1. https://codecatalyst.aws/ CodeCatalyst で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。ブループリントをフィーチャーブランチで開発した場合は、フィーチャーブランチをデフォルトブランチへの変更とマージしてください。プルリクエストを作成して、すべての変更をデフォルト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の処理 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が保存されている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、[Add to catalog] 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

> **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。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。

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が保存されている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の名前を選択し、[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公開権限の管理

プロジェクト作成中にワークフローリリースが生成された場合、カスタムブループリントの権限はデフォルトで有効になります。公開権限が有効になると、ブループリントをスペースに公開できます。権限を無効にして、ブループリントを公開できないようにすることができます。権限が無効になっていると、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。ブループリントへの新しい変更は、ブループリントの権限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権限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権限を管理するには

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公開権限を管理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. [プロジェクト公開権限] タブを選択すると、スペースのすべてのブループリントの公開権限が表示されます。
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] を選択して公開権限を変更します。権限を有効にする場合は、権限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定します。

カスタムブループリントのバージョン管理

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。ブループリントのカタログバージョンを変更しても、別のブループリント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/ 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バージョンを変更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. スペースブループリントテーブルで、管理したい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。

依存関係とツールの使用

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、各設計図のコードベースのプロジェクトモデルとして機能します。これにより、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できます。`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発信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. ファイルで`projen.ts`、deps セクションに移動します。

2. ブループリント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 の更新後、リポジトリパラメータに関する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、依存コンポーネントが古いバージョンにピン留め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。ブループリントで提供されるパッケージは、バージョンの処理方法が不明な場合を除き、同じ最新バージョン (`0.0.x`) で保管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

ブループリントでは、ツールに Yarn が使用されます。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 を使用して誤って依存関係を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必要があります。バージョンの処理方法が不明な場合は、すべてのコンポーネントを同じバージョンにしておく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保証されます。

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/*
```

モノレポを使用しない場合は、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、手動で最新バージョンに増やしますyarn。すべてのコンポーネントと util は同じバージョン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。

でのブループリントのクォータ CodeCatalyst

次の表は、Amazon のブループリントの割り当てと制限を示しています。 CodeCatalystAmazon 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

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 CodeCatalyst	100

のソースリポジトリ 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 ソースリポジトリを使用して、 CodeCatalyst プロジェクトのアセットを安全に保存、バージョン管理、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、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。

1 CodeCatalyst つのプロジェクトには複数のソースリポジトリを設定できます。たとえば、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、ドキュメント用に別々のソースリポジトリを用意したい場合があります。

ソースリポジトリ、プルリクエスト、Dev Environment のコードを扱う場合に考えられるワークフローを 1 つ紹介します。 CodeCatalyst

Mary Major は、 CodeCatalyst ブループリントを使用して Web アプリケーションプロジェクトを作成します。ブループリントは、サンプルコードを含むソースリポジトリを作成します。彼女は友人のリー・フアン、サーンヴィ・サーカー、ホルヘ・ソウザを招待して、一緒にプロジェクトに取り組んでもらいます。Li Juan はソースリポジトリ内のサンプルコードを見て、コードにテストを追加するために簡単な変更を加えることにしました。*Li ##########IDE AWS Cloud9 ##########################*開発環境が開きます。Li はすばやくコードを追加し、変更を加えたブランチをコミットして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。プルリクエストを作成する一環として、Li は Jorge Souza と Saanvi Sarkar をレビュアーとして追加し、コードのレビューを確実にします。

Jorge Souza はコードをレビューしているときに、 GitHub 作業中のアプリのプロトタイプを含む自分のプロジェクトリポジトリを持っていることを思い出します。彼は Mary Major に、 GitHub リポジトリを追加のソースリポジトリとしてプロジェクトにリンクできるようにする拡張機能をインストールして設定するように依頼します。Mary は Jorge GitHub 上のリポジトリを見直し、Jorge GitHub と協力してエクステンションを設定します。そうすれば、 GitHub 彼はそのリポジトリをプロジェクトの追加ソースリポジトリとしてリンクできるようになります。

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複製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アクセストークン (PAT) を作成して使用できます。これらの PAT はユーザー ID に関連付けられます。 CodeCatalyst詳細については、「Amazon での個人アクセストークンの管理 CodeCatalyst」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、あなたや他のプロジェクトメンバーがコードの変更を確認したりコメントしたりできる簡単な方法です。 CodeCatalyst コンソールで変更を確認したり、コード行にコメントしたりできます。

CodeCatalyst ソースリポジトリのブランチにプッシュすると、ワークフローの実行が自動的に開始され、変更のビルド、テスト、デプロイが可能になります。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は、プロジェクトの一部として 1 つ以上のワークフローが自動的に設定されます。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、「でのワークフロー入門 CodeCatalyst」を参照してください。

トピック

ソースリポジトリの概念

CodeCatalyst ソースリポジトリを扱う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

- ソースリポジトリ
- 開発環境
- パーソナルアクセストークン (PAT)
- ブランチ
- デフォルトブランチ
- コミット
- プルリクエスト
- リビジョン
- ワークフロー

プロジェクト

プロジェクトとは、 CodeCatalyst 開発チームとタスクをサポートする共同作業です。プロジェクトを作成したら、ユーザーやリソースを追加、更新、削除したり、プロジェクトダッシュボードをカスタマイズしたり、チームの作業の進捗状況を監視したりできます。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固有のものです。プロジェクト間でリポジトリを共有したり、スペース内の複数のプロジェクトにリポジトリをリンクしたりすることはできません。プロジェクトでコントリビューターまたはプロジェクト管理者のロールを持つユーザーは、そのロールに付与された権限に従って、そのプロジェクトに関連するソースリポジトリを操作できます。詳細については、「Amazon のロールを使った作業 CodeCatalyst」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、 CodeCatalyst ソースリポジトリは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は複数のソースリポジトリを設定できます。でプロジェクトのソースリポジトリを作成できます。また CodeCatalyst、インストール済みのエクステンションでそのサービスがサポートされている場合は、別のサービスがホストす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。たとえば、 GitHub GitHub Repositories エクステンションをインストールした後に、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、「でのソースリポジトリの操作 CodeCatalyst」および「クイックスタート: GitHub でのリポジトリの使用 CodeCatalyst」を参照してください。

開発環境

Dev Environment はクラウドベースの開発環境で CodeCatalyst 、これ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すばやく操作できます。Dev Environment 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にある 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 が自動的に適用されます。デフォルト 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、および 16 GiB の永続ストレージを搭載するように構成されています。

ソースリポジトリの既存のブランチを Dev Environment にクローンするか、Dev Environment 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。

パーソナルアクセストークン (PAT)

パーソナルアクセストークン (PAT) はパスワードに似ています。ユーザー ID と関連付けられ、内のすべてのスペースとプロジェクトで使用できます。 CodeCatalystPAT を使用して、統合開発環境 (IDE) や Git CodeCatalyst ベースのソースリポジトリを含むリソースにアクセスします。PAT CodeCatalyst はユーザーを代表するものであり、ユーザー設定で管理できます。1 人のユーザが複数の PAT を持つことができます。個人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、必ずローカルコンピューターに安全に保管してください。デフォルトでは、PAT は 1 年後に有効期限が切れます。

統合開発環境 (IDE) で作業する場合、PAT は Git パスワードと同等です。IDE を Git リポジトリと連携するように設定するときに、パスワードの入力を求められたら PAT を入力します。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 のマニュアルを参照してください。

ブランチ

ブランチは Git や in CodeCatalyst のコミットへのポインタまたは参照です。ブランチを使って作業を整理できます。たとえば、ブランチを使うと、他のブランチのファイルに影響を及ぼすことなく、新しいバージョンや異なるバージョンのファイルで作業できます。ブランチは新機能の開発、プロジェクトの特定のバージョンの保存などに使用できます。ソースリポジトリは 1 つのブランチを持つことも、複数のブランチを持つこともできます。テンプレートを使用してプロジェクトを作成すると、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 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。

デフォルトブランチ

CodeCatalyst のソースリポジトリには、作成方法に関係なくデフォルト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、リポジトリの作成時にデフォルトブランチが作成されます。このデフォルトブランチは 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。どのブランチをデフォルトブランチとして使用するかは変更できます。詳細については、「リポジトリのデフォルトブランチを表示および変更する」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、1 つまたは複数のファイルに対する変更です。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、変更を加えたユーザーの ID、変更の日時、コミットのタイトル、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 でのコミットの処理 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とコンテンツへの変更のスナップショットです。Git タグをコミットに追加して、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、ソースリポジトリ内のあるブランチから別のブランチへのコード変更をあなたや他のユーザーがレビュー、コメントしたり、マージしたりする主な方法です。プルリクエストを使用すると、コードの変更を共同でレビューして、軽微な変更や修正、主要な機能の追加、リリースされたソフトウェアの新しいバージョンがないかを確認することができます。プルリクエストでは、ソースブランチとターゲットブランチ間の変更や、それらのブランチのリビジョン間の違いを確認できます。コード変更の個々の行にコメントを追加できるほか、プルリクエスト全体に対するコメントも追加できます。

> **Tip**
>
> プルリクエストを作成している間、表示される違いはソースブランチの先端と宛先ブランチの先端の違いです。プルリクエストが作成されると、選択したプルリクエストのリビジョンと、プルリクエストを作成したときに宛先ブランチの先端にあったコミットの違いが表示されます。Git の違いとマージベースの詳細については、Git ドキュメントのを参照してくださいgit-merge-base。

リビジョン

リビジョンとは、プルリクエストの更新版です。プルリクエストのソースブランチにプッシュするたびに、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。ソースブランチと宛先ブランチの違いに加えて、プルリクエストのリビジョン間の違いも表示できます。詳細については、「Amazon でのプルリクエストの処理 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と継続的デリバリー (CI/CD) システムの一部としてコードをビルド、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、つまりアクションを定義します。ワークフローでは、ワークフローを開始させるイベント、つまりトリガーも定義されます。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エディターまたは YAML エディター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、ブループリントを使用してプロジェクトを作成してください。各ブループリントには、レビュー、実行、実験が可能な、機能するワークフローが導入されています。

ソースリポジトリには、プロジェクトのワークフロー、通知、課題、その他の設定情報に関する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。設定ファイルを必要とするリソースを作成したり、リポジトリをワークフローのソースアクションとして指定したりすると、設定ファイルが作成され、ソースリポジトリに保存されます。ブループリントからプロジェクトを作成する場合、設定ファイルは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すでに保存されています。この設

定情報は、.codecatalystリポジトリのデフォルトブランチにあるという名前のフォルダーに保存されます。デフォルトブランチのブランチを作成すると、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が作成されます。

ソースリポジトリを操作するためのセットアップ

CodeCatalyst ローカルマシン上の Amazon のソースリポジトリを操作する場合、Git を単独で使用することも、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たり、コードをプッシュ/プルしたりすることもできます。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。最新バージョンの Git 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 なしで 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。

個人アクセストークンを作成します。

ソースリポジトリをローカルマシンまたはお好みの IDE に複製するには、個人アクセストークン (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**
>
> 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。

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] には、デフォルトの日付をそのまま使用するか、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。有効期限のデフォルトは、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 シークレットは安全な場所に保存してください。

> **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は取得できません。

CodeCatalyst ソースリポジトリとシングルページアプリケーションブループリント入門

このチュートリアルの手順に従って、Amazon CodeCatalyst のソースリポジトリを操作する方法を学んでください。

Amazon CodeCatalyst でソースリポジトリを使い始め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、以下の方法を学ぶことができます。

- プロジェクトのソースリポジトリを表示し、その内容を参照します。
- コードを操作できる新しいブランチを含む開発環境を作成します。
- ファイルを変更し、変更内容をコミットしてプッシュします。
- プルリクエストを作成し、コードの変更を他のプロジェクトメンバーとレビューします。

- プロジェクトのワークフローを確認し、プルリクエストのソースブランチで変更を自動的にビルドしてテストします。
- ソースブランチの変更をターゲットブランチにマージし、プルリクエストを閉じます。
- マージされた変更を確認して、自動的にビルド、デプロイします。

このチュートリアルを最大限に活用するには、他の人をプロジェクトに招待して、プルリクエストで一緒に作業できるようにしてください。また、課題を作成してプルリクエストに関連付けたり CodeCatalyst、関連するワークフローが実行されたときに通知を設定してアラートを受け取るなど、その他の機能を調べることもできます。について詳しくは CodeCatalyst、を参照してください。入門チュートリアル

ブループリントを使ったプロジェクトの作成

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。ブループリン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と、コードを変更すると自動的にコードをビルド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順を追って説明しますが、ソースリポジトリがあるプロジェクトであればどのプロジェクトでも手順に従うことができます。必ず IAM ロールを選択するか、IAM ロールがない場合はプロジェクト作成時に追加してください。CodeCatalystWorkflowDevelopmentRole-*spaceName*このプロジェクトではサービスロールを使用することをお勧めします。

すでにプロジェクトがある場合は、スキップしてに進んでくださいプロジェクトのリポジトリを表示する。

> **Note**
>
> でプロジェクトを作成できるのは、スペース管理者またはパワーユーザーロールを持つユーザーだけです CodeCatalyst。このロールがなく、このチュートリアルで取り組むプロジェクトが必要な場合は、いずれかのロールを持つ人にプロジェクトを作成してもらい、作成したプロジェクトに追加してもらってください。詳細については、「Amazon のロールを使った作業 CodeCatalyst」を参照してください。

ブループリントを使っ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したい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. [設計図から始める] を選択します。
4. ブループリントを選択し、[次へ] を選択します。
5. [プロジェクトに名前を付ける] に、プロジェクトに割り当てる名前と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. 「プロジェクトリソース」で、共通のプロジェクトパラメータを設定します。
7. (オプション) 選択したプロジェクトパラメータに基づいて更新された定義ファイルを表示するには、「プロジェクトプレビューを生成」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。
8. (オプション) ブループリントのカードから [詳細を表示] を選択すると、ブループリントのアーキテクチャの概要、必要な接続と権限、ブループリントが作成するリソースの種類など、ブループリントに関する特定の詳細が表示されます。
9. [プロジェクトを作成] を選択します。

   プロジェクトブループリントの詳細については、を参照してください。プロジェクトブループリントリファレンス

プロジェクトを作成するか、プロジェクトへの招待を受け入れてサインインプロセスを完了するとすぐに、プロジェクト概要ページが開きます。新しいプロジェクトのプロジェクト概要ページには、未解決の課題やプルリクエストはありません。オプションで、課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、「での課題の作成 CodeCatalyst」および「ユーザーをプロジェクトに招待する。」を参照してください。

プロジェクトのリポジトリを表示する

プロジェクトのメンバーは、その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。GitHub スペース管理者ロールを持つユーザーがリポジトリエクステンションをインストールして設定した場合は、 GitHub GitHub エクステンション用に設定されたアカウントに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: GitHub でのリポジトリの使用 CodeCatalyst」を参照してください。

> **Note**
>
> ```
> ########################################################################## spa-app ###
> ```

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。

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に設定されている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、コードのビルド、テスト、デプロイの初期ワークフローは、完了するまでに数分かかるため、引き続き実行されます。「関連ワークフロー」の下にある番号を選択すると、関連するワークフローとそのステータスを表示できますが、これにより 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見てみましょう。`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レンダリングされます。Files には、デフォルトブランチのコンテンツが表示されます。別のブランチがある場合は、その内容を表示するようにファイルビューを変更できます。`.codecatalyst`このフォルダーには、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。

フォルダーの内容を表示するには、フォルダー名の横にある矢印を選択して展開します。たとえば、の横にある矢印を選択すると、そのフォルダーに含まれる単一ページの Web アプリケーションのファイルが表示されます。`src`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「ファイルを表示」が開き、複数のファイルの内容を参照できます。コンソールでは 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 Dev Environment を作成する必要があります。

開発環境の作成

Amazon CodeCatalyst コンソールのソースリポジトリ内の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処理するには、Dev Environment を使用するか、リポジトリをローカルコンピューターに複製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で AWS Cloud9 Dev Environment を作成します。**`develop`**別のブランチ名を選択する

こともできますが、ブランチに名前を付けると**develop**、このチュートリアルの後半でプルリクエストを作成したときに、ワークフローが自動的に実行されてコードのビルドとテストが行われます。

> **Tip**
>
> Dev Environment を使用する代わりに、またはそれに加えてローカルでリポジトリをクローンする場合は、ローカルコンピュータに Git がインストールされているか、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セットアップ」を参照してください。

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、「開発環境では統合開発環境がサポートされています。」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。

Dev Environment を作成したら、ファイルの編集、変更のコミット、**test**ブランチへの変更のプッシュを行うことができます。このチュートリアルでは、`<p>App.tsx`src`フォルダー内のファイル内のタグ間のコンテンツを編集して、Web 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ブラウズします。`src`展開して開きます`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、git commitgit pushおよびコマンドを使用して変更をコミット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には、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すると、コードの変更を共同でレビューして、軽微な変更や修正、主要な機能の追加、リリースされたソフトウェアの新しいバージョンがないかを確認できます。このチュートリアルでは、**##**。テンプレートを使用して作成し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 (存在する場合) 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 [More] を選択し、[Create pull request] 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。自分自身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。

「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」領域の情報を確認することで、このプルリクエストの作成時に開始された関連ワークフローに関する情報を確認できます。実行されたワークフローを表示するには、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**、ワークフローは自動的に実行されて変更のビルドとテストは行われません。これを設定したい場合は、onPullRequestBuildAndTestワークフローの YAML ファイルを編集してください。詳細については、「ワークフローの作成、編集、削除」を参照してください。

このプルリクエストにコメントしたり、他のプロジェクトメンバーにコメントを求めたりできます。また、任意または必須のレビュア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にさらに変更を加え、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することもできます。詳細については、、プルリクエストのレビュープルリクエストの更新Amazon でのプルリクエストの処理 CodeCatalyst、ワークフロー実行のステータスと詳細の表示およびを参照してください。

プルリクエストをマージする

プルリクエストがレビューされ、必要なレビュアーから承認を受けたら、コンソールでソースブランチをターゲットブランチにマージできます。 CodeCatalystプルリクエストをマージすると、移行先ブランチに関連するすべてのワークフローを通じて変更の実行が開始されます。このチュートリアルでは、テストブランチを main にマージして、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 [Merge] を選択します。
2. 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。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は表示されなくなります。デフォルトビューには、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、承認したりステータスを変更したりすることはできません。

> **Note**
>
> Merge ボタンがアクティブになっていないか、Not mergeable というラベルが表示されている場合は、必須のレビュー担当者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ないかのどちらかです。 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ても Merge ボタンがアクティブにならない場合は、マージコンフリクトが発生しているか、スペースのストレージ容量を超えている可能性があります。Dev Environment 内のターゲットブランチのマージコンフリクトを解決し、変更をプッシュしてからプルリクエストをマージすることも、コンフリクトを解決してローカルでマージし、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。 CodeCatalyst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。

デプロイされたコードを表示する

次は、デフォルトブランチにあった元々デプロイされたコードと、自動的にビルド、テスト、デプロイされた変更をマージして確認してみましょう。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] の `onPushToMainDeployPipeline` [最近の実行] を展開します。

> **Note**
>
> これは、シングルページ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のワークフローのデフォルト名です。

2. 最新の実行は、mainブランチへのマージされたプルリクエストコミットによって開始されたもので、ステータスが「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すると、その実行の詳細が開きます。
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 Web アプリケーションの URL です。新しいブラウザータブを開いて値を貼り付けると、ビルドされデプロイされたコードが表示されます。タブは開いたままにしておきます。
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。完了したら、開いたタブに戻って Web アプリケーションを表示し、ブラウザーを更新します。マージしたプルリクエストで行った変更が表示されるはずです。

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操作方法を調べたら、必要のないリソースは削除したくなるかもしれません。プルリクエストは削除できませんが、クローズすることはできます。作成したブランチはどれでも削除できます。

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、「ソースリポジトリを削除する」および「Amazon でプロジェクトを削除する CodeCatalyst」を参照してください。

でのソースリポジトリの操作 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や通知に関する設定ファイルやその他の情報も含まれます。この構成情報は、.codecatalyst という名前のフォルダーに保存されます。

ソースリポジトリは、 CodeCatalyst プロジェクト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でプロジェクトを作成するか、既存のプロジェクトにソースリポジトリを作成することで作成できます。プロジェクトユーザーには、プロジェクト用に作成したリポジトリが自動的に表示され、使用できるようになります。ホストされている Git GitHub 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。そうすると、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にある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はリンクできません。空のリポジトリをリンクすることはできますが、初期コミットで初期化し、 CodeCatalyst デフォルトブランチを作成するまでは使用できません。詳細については、「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。プロジェクトに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やフォルダーを表示、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処理するには、指定したリポジトリを複製する開発環境を作成してそこに分岐し、開発環境用に選択した統合開発環境 (IDE) でコードを操作できます。ローカルコンピューター上のソースリポジトリを複製し、ローカルリポジトリと内のリモートリポジトリ間で変更を取り込んだりプッシュしたりできます。 CodeCatalystIDE が認証情報管理をサポートしていれば、使用する IDE でソースリポジトリへのアクセスを設定してソースリポジトリを操作することもできます。

CodeCatalyst リポジトリ名はプロジェクト内で一意でなければなりません。

トピック

- ソースリポジトリの作成
- ソースリポジトリをリンクする
- ソースリポジトリを表示する
- ソースリポジトリの設定を編集する
- ソースリポジトリのクローニング
- ソースリポジトリを削除する

ソースリポジトリの作成

Amazon で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、 CodeCatalyst 自動的にソースリポジトリが作成されます。そのソースリポジトリには、作成した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。のリポジトリを使い始めるには、この方法が推奨されます。 CodeCatalystプロジェクト用のリポジトリを作成することを選択できます。これらのリポジトリには、いつでも編集または削除できる README.md ファイルという 1 つのファイルが含まれます。ソースリポジトリを作成するときの選択によっては、リポジトリにもファイルが含まれる場合があります。`.gitignore`

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。

ソースリポジトリをリンクする

ソースリポジトリをプロジェクトにリンクするときに、 CodeCatalyst リポジトリをホストするサービスのエクステンションが自分のスペースにインストールされている場合は、そのエクステンションを含むリポジトリを含めることができます。スペース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。エクステンションをインストールすると、そのエクステンションによってアクセスが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、「」での拡張機能のインストールとアンインストール CodeCatalyst または「」をフォローしてくださいクイックスタート: GitHub でのリポジトリの使用 CodeCatalyst。

> **Note**
>
> リポジトリはスペース内の 1 つのプロジェクトにのみリンクできます。アーカイブされたリポジトリはリンクできません。空のリポジトリをリンクすることはできますが、初期コミットで初期化し、 CodeCatalyst デフォルトブランチを作成するまでは使用できません。

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したい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、スペース管理者ロールを持つユーザは、まずリポジトリをホストするプロバイダ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、「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. [リポジトリを追加] を選択し、[リポジトリをリンク] を選択します。
4. 「リポジトリプロバイダー」で、プロバイダーの名前を選択します。GitHub account では、リポジトリでの作業に使用する連結アカウントを選択し、次に [Repository] でリポジトリの名前を選択します。

   > **Note**
   >
   > リンクできるのはアクティブなリポジトリだけです。アーカイブされたリポジトリはリンクできません。

5. 情報を確認し、[リンク] を選択します。

> **Note**
>
> リポジトリはスペース内の 1 つのプロジェクトにのみリンクできます。

リポジトリをプロジェクトからリンク解除するには、スペース管理者ロールが必要です。詳細については、「GitHub でのリポジトリの管理 CodeCatalyst」を参照してください。

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。

ソースリポジトリの設定を編集する

リポジトリの説明の編集、デフォルトブランチの選択、ブランチルールの作成と管理、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で管理できます CodeCatalyst。これにより、プロジェクトメンバーがリポジトリの用途を理解しやすくなり、チームが使用するベストプラクティスやプロセスを実施しやすくなります。

> **Note**
>
> ソースリポジトリの名前は編集できません。
> では、リンクされているリポジトリの名前、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で情報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、設定を編集したい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、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。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 [その他] 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を表示および変更する」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権限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 [編集] を選択します。詳細については、「ブランチの許可されたアクションをブランチルールで管理」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「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を管理」を参照してください。

ソースリポジトリのクローニング

ソースリポジトリ内の複数のファイル、ブランチ、コミットを効果的に処理するには、ソースリポジトリをローカルコンピューターに複製し、Git クライアントまたは統合開発環境 (IDE) を使用して変更を加えます。Issue CodeCatalyst やプルリクエストなどの機能を処理するには、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。また、コードを扱うための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 Dev Environment に自動的に複製されます。

> **Note**
>
> CodeCatalyst リンクされたリポジトリをコンソールで複製したり、そのリポジトリの Dev Environment 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にクローンするには、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、クローニングします。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/ CodeCatalyst で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. コードを編集したい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、「開発環境では統合開発環境がサポートされています。」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。
   - 「Work in new branch」を選択し、「Branch name」フィールドにブランチ名を入力し、「Create branch from」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 Dev Environment の名前を追加するか、その設定を編集します。
8. [作成] を選択します。

ソースリポジトリをクローンするには

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、[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。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。
3.
4. [リポジトリをクローン] 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**
>
> 個人アクセストークン (PAT) がない場合は、[Create token] を選択します。トークンをコピーし、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) から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。

   - リポジトリをローカルコンピューターに複製するには、ターミナルまたはコマンドラインを開き、コマンドの後にクローン URL を付けてコマンドを実行します。git clone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、先ほど保存した PAT を貼り付けます。

     > **Note**
     >
     > オペレーティングシステムに認証情報管理機能がある場合や、認証情報管理システムがインストールされている場合は、PAT を 1 回入力するだけで済みます。そうでない場合は、すべての Git 操作に PAT を指定する必要があるかもしれません。ベストプラクティスとして、認証情報管理システムに PAT を安全に保存するようにしてください。PAT をクローン URL 文字列の一部に含めないでください。

   - IDE を使用してリポジトリを複製するには、ご使用の IDE のドキュメントに従ってください。Git リポジトリを複製するオプションを選択し、URL を指定します。パスワードの入力を求められたら、PAT を入力します。

ソースリポジトリを削除する

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。ソースリポジトリに依存するワークフローがある場合、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している課題は削除または変更されませんが、Issue 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機能しなくなります。

> **Important**
>
> ソースリポジトリの削除は元に戻せません。ソースリポジトリを削除すると、そのソースリポジトリをクローニングしたり、そこからデータを引き出したり、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ーのディレクトリとファイル管理ツールを使用してください。

> **Note**
>
> CodeCatalyst リンクされたリポジトリはコンソールでは削除できません。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、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 GitHub でのリポジトリの管理 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、[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 [More] を選択し、[リポジトリを削除] 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、まだ使用中のリポジトリや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てください。続行する場合は、「delete」と入力し、「Delete」を選択します。

Amazon ンの支店との連携 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。ブランチは、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に使用できます。ソースリポジトリのブ

ランチにルールを設定して、ブランチでの特定のアクションをそのプロジェクト内の特定のロールに制限できます。

Amazon CodeCatalyst のソースリポジトリには、作成方法に関係なく、コンテンツとデフォルト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ブランチやコンテンツがない場合でも、 CodeCatalyst 初期化してデフォルトブランチを作成するまでは使用できない場合があります。ブループリントを使用してプロジェクトを作成すると、README.md ファイル、サンプルコード、ワークフロー定義、 CodeCatalyst 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が作成されます。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、デフォルトブランチが自動的に作成されます。このデフォルトブランチは 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。また、検索ではデフォルトブランチの結果のみが表示されます。他のブランチのコードを検索することはできません。

CodeCatalyst でリポジトリを作成すると最初のコミットも作成され、README.md ファイルが含まれたデフォルト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 main 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

ブランチの作成と削除

CodeCatalyst コンソールを使用して、 CodeCatalyst リポジトリ内のブランチを作成したり削除したりできます。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取り出すときに表示さ

れます。ブランチを削除しても、ユーザーが変更を取り出して同期するまで、そのブランチのコピーはローカルコンピューター上のリポジトリのクローンに残ります。

> **Tip**
>
> また、Dev Environment の作成の一環としてブランチを作成してコードを操作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したり削除したりすることもできます。詳細については、「ブランチ用の一般的な Git コマンド」を参照してください。

ブランチ (コンソール) を作成するには

1. CodeCatalyst コンソールで、ソースリポジトリが置かれてい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 [More] を選択し、[Create branch] 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、[Create] を選択します。

ブランチを削除するには (コンソール)

1. リポジトリが置かれてい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、[View all] を選択します。
4. 削除するブランチを選択し、[Delete branch] 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、オープンなプルリクエストの数、ブランチに関連する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。

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

Amazon のソースリポジトリで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。 CodeCatalyst のすべてのソースリポジトリには、作成方法に関係なく、コンテンツとデフォルトブランチがあります。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は main という名前になります。デフォルトブランチの内容は、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違った扱いになります。名前の横には Default という特別なラベルが付いています。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ーにリポジトリをクローンするときに、ローカルリポジトリ (repo) 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、課題の情報を保存したりするワークフローを作成するときにもデフォルトとして使用されます。Search in を使用すると 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。デフォルトブランチはプロジェクトの多くの面で基本となるため、デフォルト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、デフォルトブランチとして別のブランチを使用することもできます。その場合、以前のデフォルトブランチに適用されてい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**
>
> プロジェクト内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. リポジトリが置かれてい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。

デフォルトブランチを含め、設定を表示したいリポジトリを選択します。

3. リポジトリの概要ページで [その他] 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 Default というラベルとともに表示されます。これと同じラベルが Branches のブランチリストのブランチ名の横に表示されます。
5. デフォルトブランチを変更するには、「編集」を選択します。

> **Note**
>
> デフォルトブランチを変更するには、プロジェクトのプロジェクト管理者ロールが必要です。

6. デフォルトブランチに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、[保存] を選択します。

ブランチの許可されたアクションをブランチルールで管理

ブランチを作成すると、そのロールの権限に基づいてそのブランチに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、プロジェクトにおけるユーザーの役割に基づいています。コミットをブランチにプッシュするなど、あらかじめ定義されたアクションの一部を、プロジェクト内の特定の役割を持つユーザーに限定することができます。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しやすくなります。たとえば、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した場合、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。

ブランチ用のルールを作成することによる影響をすべて慎重に検討する必要があります。たとえば、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限定することを選択した場合、コントリビューターロールの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、それらのユーザーは変更をコミットして YAML にプッシュすることができないからです。ベストプラクティスとして、ブランチルールは作成後にテストして、意図し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を管理」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、プロジェクト管理者ロールが必要です。 CodeCatalyst リンクされたリポジトリにはブランチルールを作成できません。
> ロールのデフォルト権限よりも制限の厳しいブランチルールしか作成できません。プロジェクト内のユーザーのロールで許可されている範囲よりも制限の厳しいブランチルールは作成できません。たとえば、レビュー担当者の役割を持つユーザーがブランチにプッシュすることを許可するブランチルールは作成できません。

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すべてのブランチに自動的に適用されます。以前デフォルトブランチとして設定されていたブランチは、削除からの保護がなくなることを除いて、適用されたルールはそのまま保持されます。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 では、ブランチで許可されるアクションは、ユーザーがブランチアクションに対して持つロールの権限と一致するアクションです。 CodeCatalyst どのロールにどの権限があるかについての詳細は、を参照してくださいAmazon のロールを使った作業 CodeCatalyst。「カスタム」とは、1 つ以上のブランチアクションに、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り、そのロールがプロジェクト内のユーザーのロールによって付与されるデフォルトの権限とは異なることを示します。

> **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と、「ブランチを削除」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。

次の表は、ブランチでのアクションの実行を許可されるロールのアクションとデフォルト設定の一覧です。

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ていない場合にこのアクションを実行できるロール
ブランチへのマージ (これには、プルリクエストをブランチにマージすることも含まれます)	プロジェクト管理者、コントリビューター
ブランチにプッシュ	プロジェクト管理者、コントリビューター
ブランチを削除する。	プロジェクト管理者、コントリビューター
ブランチ (デフォルトブランチ) を削除する	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. リポジトリが置かれてい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したいリポジトリを選択します。
3. リポジトリの概要ページで [Branches] を選択します。
4. Branch rules 列には、リポジトリの各ブランチのルールのステータスが表示されます。Standard によると、ブランチアクションのルールはソースリポジトリで作成されたすべての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権限と一致します。Custom 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確認するには、レビューしたい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、[設定を管理] 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. Branch では、ルールを設定したい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。許可されているアクションタイプごとに、そのアクションの実行を許可するロールをドロップダウンリストから選択し、[保存] を選択します。

ブランチ用 Git コマンド

Git を使用して、コンピューターにあるソースリポジトリのクローン (ローカルリポジトリ) または Dev Environments でブランチを作成、管理、削除し、 CodeCatalyst 変更をコミットしてソースリポジトリ (リモートリポジトリ) にプッシュできます。例:

ブランチ用の一般的な 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既存のすべてのブランチに関する情報をローカルリポジトリに取り込みます。	`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 repo 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`

指定したブランチ名を使用してローカル repo 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`
ローカルリポジトリで指定されたリモートリポジトリの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内のブランチの上流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対して持つ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、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、`--delete`コマンドの一部として指定します。	`git push remote-name :branch-name` `git push remote-name --delete branch-name`

詳細については、Git のマニュアルを参照してください。

ブランチと詳細を表示する

ファイル、フォルダ CodeCatalyst、特定のブランチの最新のコミットなど、Amazon のリモートブランチに関する情報を Amazon CodeCatalyst コンソールで表示できます。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に関する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、ブランチを表示したい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] を選択してから、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。

   ブランチを表示したい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ーのリスト、最新のコミットに関する情報、README.md ファイル (ブランチに存在する場合) の内容が表示されます。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そのブランチを選択します。
4. リポジトリのすべてのブランチを表示するには、[View all] 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、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や詳細を表示する方法については、「ブランチ用の一般的な 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。

Amazon でのファイルの操作 CodeCatalyst

Amazon では CodeCatalyst、ファイルは、ファイルが保存されているソースリポジトリおよびブランチのユーザー自身や他のユーザーが利用できる、バージョン管理された自己完結型の情報です。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ファイルにコミットされた変更をすべて自動的に追跡します。1 つの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。

ソースリポジトリ内の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
- ファイルを表示する
- 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

ファイルを作成してソースリポジトリに追加するには、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。このエディターは、リポジトリ内のブランチにある README.md ファイルなどの単純なファイルを作成または編集する場合に便利です。複数のファイルを扱う場合は、開発環境を作成すること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/ CodeCatalyst で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. コードを編集したい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、「開発環境では統合開発環境がサポートされています。」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。
   - 「Work in new branch」を選択し、「Branch name」フィールドにブランチ名を入力し、「Create branch from」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 Dev Environment の名前を追加するか、その設定を編集します。
8. [作成] を選択します。

コンソールでファイルを作成するには CodeCatalyst

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を表示する。
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) デフォルトブランチとは別の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. [ファイルを作成] を選択します。
5. [ファイル名] にファイルの名前を入力します。エディター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ーまたはサブディレクトリにファイルを作成する場合は、その構造をファイル名の一部に含めてください。

   変更に問題がなければ、「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、必要に応じて変更を加えます。オプションで、Branch 内の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」に、この変更を行った理由を簡潔かつわかりやすく説明するオプションを入力します。これは、ソースリポジトリにファイルを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ーにコピーし、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 Dev Environment を使用してコマンドを実行する必要があります。git submodule add 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、プルリクエストの Git サブモジュールの違いを確認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 の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ている統合開発環境 (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ーにクローニングします。詳細については、「ソースリポジトリのクローニング」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、ファイルをローカルリポジトリにコピーします。
3. 以下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、git add追加するファイルの名前を指定してコマンドを実行します。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コマンドの後に単一ピリオドまたは二重ピリオドを入力して、現在のディレクトリレベルでのすべての変更を含めるか (単一ピリオド)、現在のディレクトリとすべてのサブディレクトリのすべての変更を含めるか (二重ピリオド) を指定します。変更をコミットするには、git commit -mコマンドを実行してコミットメッセージを指定します。変更を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 のマニュアルとを参照してくださいブランチ用 Git コマンド。
   - Dev Environment または IDE を使用している場合は、IDE でファイルを作成して追加し、変更をコミットしてプッシュします。詳細については、IDE の開発環境 CodeCatalyst のマニュアルを参照または参照してください。

ファイルを表示する

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。デフォルトブランチと他のブランチにあるファイルを表示できます。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. ファイルを表示したいプロジェクトに移動します。詳細については、「ソースリポジトリを表示する」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。

   ファイルを表示したいリポジトリを選択します。
3. デフォルト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paper アイコンで示され、フォルダはフォルダアイコンで示されます。
4. 次のいずれかを実行します。

   - 別のブランチにあるファイルやフォルダを表示するには、ブランチのリストから目的のブランチを選択します。
   - フォルダを展開するには、一覧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、一覧から選択します。ファイルの内容がブランチに表示されます。別のブランチにあるファイルの内容を表示するには、ブランチセレクター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、「ファイルを表示」から表示するその他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。

コンソールには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ーに複製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**
>
> Git CodeCatalyst サブモジュールはコンソールに表示できません。Git サブモジュールの詳細については、Git の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、リポジトリを複製し、Git クライアントまたは統合開発環境 (IDE) を使用して変更を加えます。詳細については、の開発環境 CodeCatalystまたはソースリポジトリのクローニングを参照してください。

コンソールでファイルを編集するには CodeCatalyst

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を表示する。
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、作業したい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. エディターでファイルの内容を編集し、[Commit] を選択します。オプションで、[変更をコミット] に、変更に関する詳細情報をコミットメッセージに追加します。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

ファイルの名前変更や削除は、開発環境、ローカルのコンピュータ、または統合開発環境 (IDE) で行えます。ファイルの名前を変更または削除したら、そ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 コンソールでは、ファイルの名前を変更したり削除したりすることはできません。

Amazon でのプルリクエストの処理 CodeCatalyst

プルリクエストは、あなた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、コメントしたり、マージしたりできる主な方法です。プルリクエストを使用すると、コードの変更を共同でレビューして、軽微な変更や修正、主要な機能の追加、リリースされたソフトウェアの新しいバージョンなどを確認できます。Issue を使用してプロジェクトの作業を追跡する場合、特定の課題をプルリクエストにリンクすると、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しやすくなります。プルリクエストを作成、更新、コメント、マージ、クローズすると、プルリクエストの作成者と、プルリクエストの必須または任意のレビュアーにメールが自動的に送信されます。

> **Tip**
>
> どのプルリクエストイベントについてメールを受信するかは、プロファイルの一部として設定できます。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 つはレビューしたい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宛先ブランチです。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**
>
> プルリクエストを作成している間、表示される違いはソースブランチの先端と宛先ブランチの先端の違いです。プルリクエストを作成すると、選択したプルリクエストのリビジョンと、プルリクエストを作成したときに宛先ブランチの先端にあったコミットの違いが表示されます。Git の違いとマージベースの詳細については、Git ドキュメントのを参照してくださいgit-merge-base。

プルリクエストは特定のソースリポジトリとブランチに対して作成されますが、プロジェクトでの作業の一環として、それらを作成、表示、レビュー、閉じることができます。プルリクエストを表示して処理するために、ソースリポジトリを表示する必要はありません。プルリクエストは、作成時にステータスが 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が Merged に変更されるか、Close (状態が Closed) に変わるまで未解決のままです。

コードがレビューされたら、プルリクエストの状態を以下のいずれかの方法で変更できます。

- CodeCatalyst プルリクエストをコンソールにマージします。プルリクエストのソースブランチのコードはターゲット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 Closed に変わり、ソースブランチのコードはターゲットブランチにマージされません。

プルリクエストを作成する前に、次の操作を実行します。

- レビューしたい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。
- プロジェクトの通知を設定して、プルリクエストを作成したときに実行されるワークフローについて他のユーザーに通知できるようにします。(このステップは省略可能ですが、推奨されます)。

トピック

- プルリクエストのレビュー
- プルリクエストの更新
- プルリクエストをマージする
- プルリクエストを閉じる

プルリクエストの作成

Amazon CodeCatalyst のジェネレーティブ AI 機能はプレビューリリース中であり、変更される可能性があります。これらは米国西部 (オレゴン) リージョンでのみ利用できます。ジェネレーティブ AI 機能へのアクセスは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。

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ら、ソースブランチのコンテンツ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、プルリクエスト作成ページに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、プルリクエストがマージされたりクローズされたりしたときだけでなく、プルリクエストの更新に関するメールも自動的に届きます。

> **Note**
>
> プルリクエストを作成している間、表示される違いはソースブランチの先端と宛先ブランチの先端の違いです。プルリクエストが作成されると、選択したプルリクエストのリビジョンと、プルリクエストを作成したときに宛先ブランチの先端にあったコミットの違いが表示されます。Git の違いとマージベースの詳細については、Git ドキュメントのを参照してくださいgit-merge-base。

プルリクエストを作成するときに Write description for me 機能を使用して、プルリクエストに含まれる変更の説明を Amazon Q に自動的に作成させることができます。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宛先ブランチと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を最も適切に解釈した結果を作成します。

> **Note**
>
> Amazon Bedrock を搭載: AWS 不正使用検知を自動化しています。「自分用に説明を書く」機能と「コンテンツ概要を作成」機能は Amazon Bedrock で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、および人工知能 (AI) の責任ある使用を実現できます。

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 [More] を選択し、[Create pull request] 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、「必要な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。

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 [Source repositories] を選択し、レビューするコード変更があるブランチを含むリポジトリを選択します。

3. 既定のブランチ名の横にあるドロップダウン矢印を選択し、一覧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、[View all] を選択します。
4. [その他] を選択し、[プルリクエストを作成] を選択します。
5. リポジトリとソースブランチはあらかじめ選択されています。「ターゲットブランチ」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は、レビューすべき内容とその理由を他のプロジェクトユーザーが理解しやすいタイトルを入力します。必要に応じて、関連する課題へのリンクを貼り付けたり、加えた変更の説明を追加したりするなど CodeCatalyst、プルリクエストの説明に詳細情報を入力します。

   > **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、プルリクエストの宛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. ブランチ間の違いを確認してください。変更が表示されない場合は、ブランチが同じか、ソースとデスティネーション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) [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。
8. (オプション)「必須のレビュー担当者を追加」で、「必要な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**
   >
   > レビュア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したい変更が含まれ、必要なレビュアーが含まれていることを確認したら、[作成] を選択します。

ブランチがプルリクエストの宛先ブランチと一致する場所で実行するように設定されているワークフローがある場合は、プルリクエストの作成後に、プルリクエストの詳細領域の「概要」に実行された

ワークフロー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。

プルリクエストを表示する

Amazon CodeCatalyst のジェネレーティブ AI 機能はプレビューリリース中であり、変更される可能性があります。これらは米国西部 (オレゴン) リージョンでのみ利用できます。ジェネレーティブ AI 機能へのアクセスは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。

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、プロジェクトに関する未解決のプルリクエストがすべて表示されます。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。プルリクエストを閲覧する際、自分用に作成したプルリクエストへの変更に対して残されたコメントをすべてまとめて表示するように選択できます。

> **Note**
>
> Amazon Bedrock を搭載: AWS 不正行為の自動検出機能を実装しています。「自分用に説明を書く」機能と「コンテンツ概要を作成」機能は Amazon Bedrock で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、および人工知能 (AI) の責任ある使用を実現できます。

未解決の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したいプロジェクトに移動します。
2. プロジェクトページには、プルリクエストの作成者、プルリクエストのブランチが格納されているリポジトリ、プルリクエストの作成日などの情報を含む、未解決の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、[View all] を選択します。セレクターを使用してオプションを選択できます。たとえば、すべてのプルリクエストを表示するには、[Any status] と [Any author] 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むこともできます。

4. プルリクエストページでは、ID、タイトル、ステータスなどでプルリクエストをソートできます。プルリクエストページに表示する情報の内容と量をカスタマイズするには、ギア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、リストからそのプルリクエストを選択します。
6. このプルリクエストに関連するワークフロー実行のステータス (存在する場合) を確認するには、[概要] を選択し、[ワークフロー実行] の下にあるプルリクエストの [プルリクエストの詳細] 領域の情報を確認します。

   ワークフローにプルリクエストの作成イベントまたは改訂イベントが設定されていて、ワークフロー内の宛先ブランチの要件がプルリクエストで指定された宛先ブランチと一致する場合、ワークフローの実行が行われます。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。
7. リンクされた課題がある場合は、[概要] を選択し、[課題] の下の [プルリクエストの詳細] の情報を確認します。リンクされた課題を表示したい場合は、一覧からその 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におけるコード変更について残されたコメントの概要を作成するには、「Create content summary」を選択します。サマリーには、プルリクエスト全体に残されたコメントは含まれません。

   > **Note**
   >
   > この機能を使用するには、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。
9. プルリクエストのコード変更を確認するには、[Changes] を選択します。[Files changed] では、プルリクエストで変更されたファイルの数と、プルリクエスト内のどのファイルにコメントがあるかを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。フォルダーを展開すると、フォルダー内の各ファイルのコメント数が表示されます。コードの特定の行に残されたコメントも表示できます。

   > **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「ファイルを表示する」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、Reports を選択します。

> **Note**
>
> プルリクエストにレポートが表示されるようにするには、レポートを生成す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、「のワークフローを使用したテスト CodeCatalyst」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を管理

プルリクエストを作成すると、そのプルリクエストに必須のレビュアーを追加するか、任意のレビュアーを追加するかを選択できます。ただし、特定の宛先ブランチにマージする際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。宛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、必要なレビュアーからの承認に加えて、そのルールの要件を満たす必要があります。承認ルールを作成すると、デフォルト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すべての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には、適用されたルールが引き続き適用されます。

承認ルールを作成するときは、現在とfuture 両方で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。たとえば、プロジェクトに 6 人のユーザーがいて、移行先のブランチにマージする前に 5 件の承認を必要とする承認ルールを作成した場合、プルリクエストを作成したユーザー以外の全員に、プルリクエストをマージする前にそのプルリクエストを承認することを要求するルールを効果的に作成したことになります。

> **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、プロジェクト管理者ロールが必要です。 CodeCatalyst リンクされたリポジトリの承認ルールは作成できません。

承認ルールは削除できませんが、承認をゼロに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。

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. リポジトリが置かれてい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 [Branches] を選択します。
4. 「承認規則」列で「表示」を選択すると、リポジトリの各ブランチの規則の状況が表示されます。

   「最小承認数」の数は、プルリクエストをそのブランチにマージするまでに必要な承認数に相当します。
5. 承認ルールを作成または変更するには、[設定を管理] を選択します。ソースリポジトリの設定ページの [承認ルール] で [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. Branch では、承認ルールを設定したい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。[最小承認数] に数値を入力し、[保存] を選択します。

プルリクエストのレビュー

Amazon CodeCatalyst のジェネレーティブ AI 機能はプレビューリリース中であり、変更される可能性があります。これらは米国西部 (オレゴン) リージョンでのみ利用できます。ジェネレーティブ AI 機能へのアクセスは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。

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レビューしたりコメントしたりできます。ソースブランチとターゲットブランチの違い、またはプルリクエストのリビジョン間の違いにある個々のコード行にコメントを追加できます。プルリクエストのコード変

更について残されたコメントの概要を作成して、他のユーザーが残したフィードバックをすぐに理解できるようにすることができます。また、コードを扱うための開発環境を作成することもできます。

> **Note**
>
> Amazon Bedrock を搭載: AWS 不正行為の自動検出機能を実装しています。「自分用に説明を書く」機能と「コンテンツ概要を作成」機能は Amazon Bedrock で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、および人工知能 (AI) の責任ある使用を実現できます。

> **Tip**
>
> どのプルリクエストイベントについてメールを受信するかは、プロファイルの一部として設定できます。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、プルリクエストを作成したときに宛先ブランチの先端にあったコミットの違いを示しています。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 ドキュメントのを参照してくださいgit-merge-base。

> **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た場合は、レビューを開始する前に、ブラウザを更新してプルリクエストの最新リビジョンが利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 プルリクエストをコンソールでレビューするには

1. プロジェクトに移動します。
2. 以下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表示されている場合は、一覧から選択します。
   - プルリクエストがプロジェクトページに表示されていない場合は、「すべて表示」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探し、一覧から選択します。

- 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. レビューしたいプルリクエストをリストから選択します。プルリクエストのリストは、フィルターバーに名前の一部を入力することでフィルターできます。
4. Overview 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されたコメントを作成して表示できます。また、ワークフローの実行、リンクされている課題、レビュアー、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されたコメントは Changes 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、[プルリクエストに関するコメント] を展開し、[コメントを作成] 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における変更について残されたすべてのコメントの概要を表示するには、[コメントの概要を作成] を選択します。

> **Note**
>
> 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 機能が有効になっている必要があります。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。

7. Changes では、宛先ブランチとプルリクエストの最新リビジョンとの違いを確認できます。リビジョンが複数ある場合は、比較するリビジョンの違いを変更できます。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**
>
> 「Files changed」では、プルリクエストで変更されたファイルの数と、プルリクエスト内のどのファイルにコメントがあるかを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。フォルダーを展開すると、フォルダー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、「統一」または「分割」を選択します。
9. プルリクエスト内の行にコメントを追加するには、コメントしたい行に移動します。その行に表示されるコメントアイコンを選択し、コメントを入力して [Save] を選択します。

10. プルリクエストのリビジョン間の変更、またはソースブランチとターゲットブランチ間の変更を確認するには、比較で使用可能なオプションから選択します。リビジョン内の行のコメントは、それらのリビジョンでも保持されます。
11. プルリクエストのトリガーに関する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のラインカバレッジ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. プルリクエストのコードを変更したい場合は、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。必要に応じて Dev Environment の名前を追加するか、その設定を編集して、[Create] を選択します。
13. Reports では、このプルリクエストに含まれる品質レポートを表示できます。リビジョンが複数ある場合は、リビジョン間の差を比較するリビジョンを変更できます。レポートは名前、ステータス、ワークフロー、アクション、タイプで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するには、レポートを生成す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、「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、「のワークフローを使用したテスト CodeCatalyst」を参照してください。
15. このプルリクエストのレビュー担当者としてリストされていて、変更を承認したい場合は、最新のリビジョンが表示されていることを確認し、[承認] を選択してください。

> **Note**
>
> プルリクエストをマージする前に、必須のレビュアー全員がプルリクエストを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レビューしやすくすることができます。プルリクエストを更新して、レビュアー、Issue 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。たとえば、プルリクエストに必要なレビュアーを変更して、休暇で不在のユーザーを削除し、別のユーザーを追加したい場合があります。オープン状態のプルリクエ

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、さらにコードを変更してプルリクエストを更新することもできます。 CodeCatalyst ソースリポジトリ内のプルリクエストの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のリビジョン間の違いを確認できます。

プルリクエストのレビュアーに最新情報を伝えるには

1. プルリクエストのレビュアーを更新したい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、レビュアーに更新したい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」領域で、プラス記号を選択して必須または任意のレビュアーを追加します。レビュアーの横にある [X] を選択すると、そのレビュアーをオプションまたは必須のレビュアーとして削除できます。

4. (オプション) [概要] の [プルリクエストの詳細] 領域で [課題をリンク] を選択して課題をプルリクエストにリンクし、一覧から課題を選択するか、ID を入力します。課題のリンクを解除するには、リンクを解除したい課題の横にあるリンク解除アイコンを選択します。

プルリクエストのソースブランチにある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、Dev Environment を作成するか、リポジトリとそのソースブランチを複製し、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に変更を加えます。変更をソースリポジトリのソースブランチにコミットしてプッシュすると、プルリクエストが変更内容で自動的に更新されます。 CodeCatalyst 詳細については、「ソースリポジトリのクローニング」および「Amazon でのコミットの処理 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。 CodeCatalystコンソール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したいプロジェクトに移動します。
2. プロジェクトページには、プルリクエストの作成者、プルリクエストのブランチを含むリポジトリ、プルリクエストの作成日などの情報を含む、未解決の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別にフィルタリングできます。変更するプルリクエストをリストから選択します。

3. すべてのプルリクエストを表示するには、[View all] を選択します。または、ナビゲーションペインで [Code] を選択し、次に [Pull requests] を選択します。フィルターボックスまたはソート機能を使用して、変更したいプルリクエストを見つけて選択します。
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。

プルリクエストをマージする

コードがレビューされ、必要なレビュアー全員が承認したら、 CodeCatalyst ファストフォワードなどのサポートされているマージ戦略を使用して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 マージを評価し、コンソールに用意されているマージストラテジーと、ソースブランチをターゲットブランチにマージできるマージストラテジーのどちらかを選択できるようにします。ローカルコンピュータまたは Dev Environment git merge でコマンドを実行してソースブランチをターゲットブランチにマージすることで、プルリクエストを選択した Git マージ戦略とマージすることもできます。その後、 CodeCatalystターゲット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。

> **Note**
>
> ブランチをマージして Git に変更をプッシュしても、プルリクエストは自動的に閉じられません。

プロジェクト管理者の役割を持っている場合は、承認と承認ルールの要件をすべて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)

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なく、必要なレビュアー全員がプルリクエストを承認していれば、 CodeCatalyst コンソールでプルリクエストをマージできます。コンフリクトがあったり、マージを完了できなかったりすると、マージボタンは無効になり、「マージ不可」というラベルが表示されます。その場合は、マージする前に、必要な承認者から承認を得て、必要に応じてコンフリクトをローカルで解決し、変更内容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だけでなく、必須または任意のレビュアーにも自動的にメールが送信されます。プルリクエストにリンクされている課題が自動的にクローズされたり、ステータスが変更されたりすることはありません。

> **Tip**
>
> プロファイルの一部として、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「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 pull request (すべてのプルリクエストを表示)] を選択し、一覧から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、[Pull requests] 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」ボタンが非アクティブだったり、「マージ不可」というラベルが表示されている場合は、必須のレビュアーがまだプルリクエストを承認していないか、プルリクエストをコンソールでマージできないかのどちらかです。 CodeCatalyst プルリクエストを承認していないレビュアーは、「概要」の「プルリクエストの詳細」領域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でない場合は、マージコンフリクトが発生している可能性があります。下線の付いた「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。Dev Environment CodeCatalyst またはコンソールで宛先ブランチのマージコンフリクトを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、マージを含むコミットをソースブランチにプッシュすることもできます。 CodeCatalyst詳細については、プルリクエストのマージ (Git)および Git のマニュアルを参照してください。

マージ要件を上書きしてください。

プロジェクト管理者の役割を持っている場合は、必要な承認と承認ルールの要件をすべて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。これをプルリクエストの要件のオーバーライドと呼びます。必要なレビュアーがいない場合や、特定のプルリクエストを、すぐには満たせない承認ルールのあるブランチにマージすることが急務になった場合に選択できます。

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したいプルリクエストで、Merge ボタンの横にあるドロップダウン矢印を選択します。[承認要件の上書き] を選択します。
2. 「オーバーライド理由」に、このプルリクエストが承認ルールや必要なレビュー担当者要件を満たさないまま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、デフォルトをそのまま使用します。また、自動生成されたコミットメッセージをより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ための要件をオーバーライドする場合は、他のチームメンバーと決定を確認する機会が得られるまで、ソースブランチを保持しておくことをおすすめします。
5. [Merge (マージ)] を選択します。

プルリクエストのマージ (Git)

Git はブランチのマージと管理のための多くのオプションをサポートしています。以下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 の Web サイトで入手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 repo のソースブランチからの変更をローカル repo のターゲットブランチにマージします。	`git checkout destination-branch-name`
	
	`git merge source-branch-name`

早送りマージを指定して、ソースブランチをターゲットブランチにマージします。これによりブランチがマージされ、ターゲットブラン	`git checkout destination-branch-name`

チポインタがソースブランチの先端に移動します。	`git merge --ff-only source-branch-name`

スカッシュマージを指定して、ソースブランチをターゲットブランチにマージします。これにより、ソースブランチのすべてのコミットが宛先ブランチの 1 つのマージコミットにまとめられます。	`git checkout destination-branch-name`
	
	`git merge --squash source-branch-name`

3 ウェイマージを指定して、ソースブランチをターゲットブランチにマージします。これによりマージコミットが作成され、ソースブランチの個々のコミットがターゲットブランチに追加されます。	`git checkout destination-branch-name`
	
	`git merge --no-ff source-branch-name`

ローカル repo のソースブランチを削除します。これは、移行先ブランチにマージして変更をソースリポジトリにプッシュした後のローカルリポジトリのクリーンアップとして便利です。	`git branch -d source-branch-name`

ローカルリポジトリが指定したリモートリポジトリのニックネームを使用して、リモートリポジトリ (内のソースリポジトリ CodeCatalyst) のソース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、`--delete`コマンドの一部として指定してください。	`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 Closed としてマークできます。これによってプルリクエストがマージされるわけではありませんが、どのプルリクエストにアクションが必要で、どのプルリクエストが関連性がなくなったかを判断するのに役立ちます。プルリクエストはマージした後でクローズすることをおすすめします。プルリクエストをクローズすると、プルリクエストの作成者だけでなく、必須または任意のレビュアーにも自動的にメールが送信されます。プルリクエストにリンクされている課題のステータスは自動的に変更されません。

> **Note**
>
> プルリクエストをクローズした後で再びオープンすることはできません。

プルリクエストをクローズするには

1. プルリクエストをクローズしたいプロジェクトに移動します。
2. プロジェクトページに、未解決のプルリクエストが表示されます。クローズしたい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 でのコミットの処理 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。ユーザーがコミットして変更をブランチにプッシュするたびに、その情報が保存されます。Git のコミット情報には、コミットの作成者、変更をコミットした人、日付と時刻、加えら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、 CodeCatalyst 作成者名はユーザー名です。特定のコミットを識別しやすくするために、コミットに Git タグを追加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する。
- コミットに加えられた変更を含め、個々のコミットをその親と比較した状態で表示します。

ファイルやフォルダーも表示できます。詳細については、「Amazon でのファイルの操作 CodeCatalyst」を参照してください。

トピック

- ブランチへのコミットを表示する
- コミットの表示方法の変更 (CodeCatalystコンソール)

ブランチへのコミットを表示する

ブランチに加えられた変更の履歴は、コンソールでブランチのコミットを確認することで表示できます。 CodeCatalyst これにより、誰がいつブランチに変更を加えたかを把握できます。特定のコミットに加えられた変更を確認することもできます。

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 のマニュアル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. コミットを表示したい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したい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、そのブランチへの最新のコミットに関する情報も表示されます。[コミット] を選択します。または、[その他] を選択し、[コミットを表示] 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、ブランチセレクターを選択し、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、「Commit title」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、親コミットと指定し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することで、どの親コミットに情報を表示して変更を表示するかを選択できます。

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 などの列を非表示にするか表示するかを選択できます。

コミットの表示方法を変更するには (コンソール)

1. コミットを表示したい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したい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、そのブランチへの最新のコミットに関する情報も表示されます。[コミット] を選択します。
4. 歯車アイコンを選択します。
5. 環境設定で、表示するコミットの数を選択し、コミットの作成者、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**
   >
   > 情報を表示する際にコミットの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、[Save] を選択して変更を保存するか、[Cancel] を選択して変更を破棄します。

のソースリポジトリのクォータ CodeCatalyst

次の表は、Amazon のソースリポジトリのクォータと制限を示しています。 CodeCatalystAmazon 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

リソース	情報
	• 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 の内部構造」と git-check-ref-format「」を参照してください。
プルリクエストへのコメント	1 つのプルリクエストには最大 1,000 件です。
コミットメッセージ	最大 1024 文字です。

リソース	情報

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 の標準を使用してください。

ファイルサイズ	CodeCatalystコンソールを使用する場合、1 つのファイルにつき最大 6 MB です。
コンソールに表示できるファイルサイズ CodeCatalyst	CodeCatalystコンソールを使用する場合、1 つのファイルにつき最大 6 MB です。

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、小さなコミットを複数行うことを検討してください。
コミットのメタデータ	1 つのコミットのメタデータを組み合わせた最大サイズは 6 MB です (たとえば、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)。 **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。
CodeCatalyst プルリクエストにリンクできる課題の数	50
プルリクエストにリンクできる Jira 課題の数	50
1 スペース内のオープンプルリクエストの数	Amazon CodeCatalyst スペースは最大1,000です。

リソース	情報
1 スペース内のプルリクエストの合計数	Amazon CodeCatalyst スペースは最大10,000です。
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 スペースは最大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、ダッシュを自由に組み合わせることができます。名前は大文字と小文字を区別しません。リポジトリ名の末尾を .git にしたり、スペースを使用したり、次の文字を使用したりす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、スペースの全体的な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アー	プルリクエストのレビュアーは合計 100 人まで (オプションまたは必須)。

リソース	情報
プルリクエストの要約を書面で作成する	プルリクエストの書面による要約の最大数は、スペースの請求レベル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。

の開発環境 CodeCatalyst

開発環境はクラウドベースの開発環境です。Amazon では CodeCatalyst、Dev Environments 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場合、いくつかの選択肢があります。

- CodeCatalyst プロジェクト固有の開発環境を作成して、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 Dev Environment を作成し、ソースリポジトリからコードを複製して、サポートされている IDE でそのコードを処理します。
- IDE で開発環境を作成し、ソースリポジトリを開発環境に複製します。

開発ファイルは、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たものです。その結果、開発環境のセットアップ、プロジェクトの切り替え、開発環境設定のチームメンバーへの複製を迅速に行うことができます。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開発ファイルを使用するため、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ることができます。

Dev Environment 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にある 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 が自動的に適用されます。このデフォルト 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。ブループリントを使用してプロジェクトを作成した場合、開発ファイルはによって自動的に作成されます。CodeCatalyst開発ファイルの詳細については、https://devfile.io を参照してください。

開発環境を作成すると、その環境にアクセスできるのは自分だけです。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作業できます。

デフォルトでは、開発環境は 2 コアのプロセッサ、4 GB の RAM、16 GB の永続ストレージを搭載するように設定されています。スペース管理者権限を持っている場合は、スペースの請求範囲を変更してさまざまな Dev Environment 設定オプションを使用したり、コンピューティングとストレージの制限を管理したりできます。開発環境で考えられるワークフローは 1 つです。

Akua MansaはExample Corpの新しい開発者で、Akuaはチームの製品の新バージョンがリリースされる直前に新しいチームに加わりました。Akua のチームは、次期バージョンの製品の新機能に速やかに取り組んでほしいと彼女に求めています。面倒なセットアッププロセスを避けるため、Akua は

チームのプロジェクトへの招待を受け入れます。 CodeCatalyst 次に、関連する課題を自分自身に割り当て、チームのソースリポジトリの既存のブランチから Dev Environment を作成します。Akua の開発環境は、選択した IDE でソースリポジトリのコードを開きます。開いた IDE インスタンスは、自動的に識別され適用された devfile を含む彼女の開発環境に接続されます。開発ファイルには、彼女が作業を開始するのに必要なツールがすべて指定されています。Akua は新製品機能のコードを記述し、コードの変更をコミットし、変更を既存のブランチにプッシュし、作業が完了したら Dev Environment を削除します。Akua は、時間のかかるセットアッププロセスなしに、新しいチームのソースリポジトリにコードを提供しました。

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」、「開発環境」、または「 CodeCatalyst 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を使用して開発環境を作成します。
- 「開発環境」ページから、 CodeCatalyst ソースリポジトリに接続されていない空の開発環境を作成します。
- 任意の IDE で開発環境を作成し、任意のソースリポジトリを開発環境に複製します。

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。 CodeCatalyst 作成した開発環境は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どの IDE でも作業できます。IDE で開発環境を使用するには、AWS Toolkit をインストールしておく必要があります。詳細については、「開発環境では統合開発環境がサポートされています。」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た Dev Environment を作成した場合、Resource 列には常に Dev Environment の作成時に指定したブランチが表示されます。これは、別のブランチを作成したり、Dev Environment 内の別のブランチに切り替えたり、追加のリポジトリをクローンしたりした場合も同様です。空の Dev Environment を作成した場合、「リソース」列は空白になります。

開発環境では統合開発環境がサポートされています。

開発環境は、以下のサポートされている統合開発環境 (IDE) で使用できます。

- AWS Cloud9
- JetBrains IDE
  - IntelliJ アイデアアルティメット
  - GoLand
  - PyCharmプロフェッショナル
- Visual Studio Code

での開発環境の作成 CodeCatalyst

で開発環境を使い始めるには CodeCatalyst、AWS ビルダー ID または SSO を使用して認証を行い、サインインしてください。

ブランチから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. 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、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、「開発環境では統合開発環境がサポートされています。」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で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」に、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。
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 接続の開発環境はサポートしていないことにも注意してください。 AWS Toolkit

    > **Note**
    >
    > VPC 接続を使用して開発環境を作成すると、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`

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/ 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. 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。
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、「開発環境では統合開発環境がサポートされています。」を参照してください。
6. 「空の開発環境を作成」を選択します。
7. (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。
8. (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。
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 接続の開発環境はサポートしていないことにも注意してください。 AWS Toolkit

   > **Note**
   >
   > VPC 接続を使用して開発環境を作成すると、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`

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**
>
> Dev Environment が IDE で開いたら、コード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しなければならない場合があります。

IDE での開発環境の作成

サポートされている統合開発環境 (IDE) を使用して、プロジェクトのソースリポジトリに保存されているコードを操作できます。

- Amazon CodeCatalyst との協力 AWS Cloud9
- CodeCatalyst VS コードのAmazon

  認証の手順については、 AWS Toolkit for Visual Studio Code ユーザーガイドの「AWS 開発環境からの認証と接続」と「Amazon CodeCatalyst の認証」を参照してください。
- Amazon CodeCatalyst イン JetBrains

  認証の手順については、『 AWS Toolkit for JetBrains ユーザーガイド』の「 JetBrains Gateway を認証して接続する」を参照してください。 CodeCatalyst

開発環境の停止

`/projects`開発環境のディレクトリには、ソースリポジトリから取得したファイルと、開発環境の設定に使用される devfile が格納されます。Dev Environment /home の作成時には空のディレクトリには、Dev Environment の使用中に作成したファイルが格納されます。Dev Environment / `projects` /home のおよびディレクトリにあるものはすべて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、Dev Environment での作業を停止できます。

> **Warning**
>
> Web ブラウザ、リモートシェル、IDE 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、接続しているすべてのインスタンスを必ず閉じてください。

開発環境は、開発環境の作成時に「タイムアウト」フィールドで選択した時間アイドル状態になると、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。開発環境を作成したときに「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。その代わり、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すると、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/ CodeCatalyst でコンソールを開きます。
2. 開発環境を停止したいプロジェクトに移動します。
3. ナビゲーションペインで [Code] を選択します。
4. 「開発環境」を選択します。
5. 停止したい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**
>
> コンピュート使用量は開発環境の実行中のみ請求されますが、ストレージ使用量は開発環境が存在する期間全体にわたって請求されます。コンピュート課金を停止するには、使用していない開発環境を停止してください。

開発環境の再開

`/projects`開発環境のディレクトリには、ソースリポジトリから取得したファイルと、開発環境の設定に使用される devfile が格納されます。Dev Environment `/home` の作成時には空のディレクトリには、Dev Environment の使用中に作成したファイルが格納されます。Dev Environment `/projects` `/home` のおよびディレクトリにあるものはすべて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生じた場合は、Dev Environment での作業を停止して、後で開発環境での作業を再開できます。

開発環境は、開発環境の作成中に「タイムアウト」フィールドで選択した時間アイドル状態になると、自動的に停止します。Dev Environment AWS Cloud9 をアイドル状態にするには、ブラウザー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使用可能で稼働しています。ブランチを削除した Dev Environment での作業を再開したい場合は、新しいブランチを作成し、変更内容をそのブランチにプッシュしてください。

概要ページから開発環境を再開するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. 開発環境を再開したいプロジェクトに移動し、「My Dev Environments」セクションに移動します。
3. 「再開 (IDE)」 を選択します。
   - JetBrains IDE では、 JetBrains -gateway リンクを開くアプリケーションを選択するように求められたら Gateway-EAP を選択します。 JetBrainsプロンプトが表示されたら、「リンクを開く」を選択して確定します。
   - VS Code IDE では、VS Code リンクを開くアプリケーションの選択を求めるメッセージが表示されたら、VS Code を選択します。[リンクを開く] 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/ CodeCatalyst でコンソールを開きます。
2. 開発環境を再開したいプロジェクトに移動します。
3. ナビゲーションペインで [Code] を選択します。
4. [ソースリポジトリ] を選択します。
5. 再開したい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 IDE では、「このサイトで Gateway で JetBrains-gateway リンクを開くことを許可しますか?」というメッセージが表示されたら、「リンクを開く」を選択して確認します。JetBrains 。
- VS Code IDE では、「このサイトが Visual Studio Code で VS Code リンクを開くことを許可しますか?」というメッセージが表示されたら、[リンクを開く] を選択します。 。

「開発環境」ページから開発環境を再開するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. 開発環境を再開したいプロジェクトに移動します。
3. ナビゲーションペインで [Code] 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 IDE では、「このサイトで Gateway との JetBrains-gateway リンクを開くことを許可しますか?」というメッセージが表示されたら、「リンクを開く」を選択して確認します。JetBrains 。
- VS Code IDE では、「このサイトが Visual Studio Code で VS Code リンクを開くことを許可しますか?」というメッセージが表示されたら、[リンクを開く] を選択します。 。

> **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。コンピュート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 Dev Environment が再起動します。

開発環境を編集するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. 開発環境を編集したいプロジェクトに移動します。
3. ナビゲーションペインで [Code] 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. コンピュートタイムアウト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処理が終了したら、開発環境を削除できます。新しいコンテンツに取り組むために、新しい開発環境を作成します。Dev Environment を削除すると、保存されているコンテンツは完全に削除されます。Dev Environment を削除する前に、コード変更を Dev Environment の元のソースリポジトリにコミットしてプッシュしてください。Dev Environment を削除すると、Dev Environment のコンピューティングとストレージへの課金は停止します。

Dev Environment 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ことがあります。ストレージクォータに達すると、その間は新しい開発環境を作成できなくなります。

> **Important**
>
> 開発環境を削除すると元に戻すことはできません。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/ CodeCatalyst でコンソールを開きます。
2. 開発環境を削除したいプロジェクトに移動します。
3. ナビゲーションペインで [Code] 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] をクリックします。
7. **delete**と入力して Dev 環境の削除を確定します。
8. [削除] をクリックします。

> **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必ず削除してください。
> 開発環境を削除しても、VPC のネットワークインターフェースは削除され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 Dev Environment 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てから削除する必要があります。

SSH 経由で開発環境に接続する

SSH 経由で開発環境に接続して、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 IDE の使用などのアクションを制限なく実行できます。

> **Note**
>
> IDE のタブまたはウィンドウを閉じた後も SSH を長時間使用したい場合は、IDE での操作がないことが原因で開発環境が停止しないように、必ず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

- 以下のオペレーティングシステムのいずれかが必要です。
  - 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
  - `dpkg`、`rpm`またはパッケージマネージャーと Bash バージョン 3 `yum` 以降を搭載した Linux
- AWS CLI バージョン 2.9.4 以降も必要です。

SSH 経由で開発環境に接続するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. SSH 経由で開発環境に接続したいプロジェクトに移動します。
3. ナビゲーションペインで [Code] を選択します。
4. 「開発環境」を選択します。

5. SSH 経由で接続したい実行中の開発環境を選択します。
6. [SSH 経由でConnect] を選択し、目的のオペレーティングシステムを選択して、次の操作を行います。

   - まだ行っていない場合は、指定したターミナルに最初のコマンドを貼り付けて実行します。このコマンドはスクリプトをダウンロードし、ローカル環境で以下の変更を実行します。これにより、SSH 経由で Dev Environment に接続できるようになります。
     - のセッションマネージャー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、 AWS Config CodeCatalyst ローカルを変更して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 経由で開発環境に接続するために必要な設定を追加します。
     - SSH `~/.aws/codecatalyst-dev-env` 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 AWS Systems 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。このセッションは、ローカル SSH クライアントがリモート Dev Environment に安全に接続するために使用されます。
   - 2 CodeCatalyst 番目のコマンドを使用して 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`ディレクトリ内のスクリプトが API を呼び出せるように、認証情報をリクエストして取得します。CodeCatalyst StartDevEnvironmentSession 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、このステップの指示どおりに SSO ログインを実行していないと、エラーが発生します。
   - 3 番目のコマンドを使用して、指定した開発環境に SSH 経由でConnect します。このコマンドの構造は以下のとおりです。

     ```
     ssh codecatalyst-dev-env=<space-name>=<project-name>=<dev-environment-id>
     ```

     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とダウンロードなど、SSH クライアントで許可されている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へのファイルのアップロード:

```
scp -O </path-to-local-file> codecatalyst-dev-env=<space-name>=<project-name>=<dev-environment-id>:</path-to-remote-file-or-directory>
```

開発環境の設定

devfile はチーム全体で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。開発ファイルは、必要な開発ツールをコード化する YAML ファイルです。開発ファイルを設定することで、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 CodeCatalyst がそれらを開発環境にインストールします。devfile は作成対象のリポジトリ固有のもので、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、開発ファイル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、 CodeCatalyst 開発ファイルが自動的に作成されます。Dev Environment `/projects` のディレクトリには、ソースリポジトリと devfile から取得したファイルが格納されます。Dev Environment /home を初めて作成したときは空だったディレクトリには、Dev Environment の使用中に作成したファイルが格納されます。Dev Environment `/projects` /home のおよびディレクトリーにあるものはすべて永続的に保存されます。

> **Note**
>
> devfile または devfile コンポーネントの名前を変更した場合にのみ、/homeフォルダーが変更されます。devfile または devfile の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、/home以前のディレクトリデータを復元することはできません。

ルートに devfile を含まないソースリポジトリで Dev Environment を作成した場合、またはソースリポジトリなしで Dev Environment を作成した場合、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 IDE で同じデフォルトのユニバーサル devfile イメージが使用されます。 CodeCatalyst 現在 devfile バージョン 2.0.0 をサポートしています。[devfile について詳しくは、「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 接続の開発環境は、次の devfile イメージのみをサポートしていることに注意してください。

- ユニバーサルイメージ
- プライベート Amazon ECR イメージ (リポジトリが VPC と同じリージョンにある場合)

トピック

内の開発環境のリポジトリ開発ファイルの編集 CodeCatalyst

リポジトリ devfile を編集するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. devfile を編集したい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 [Code] 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、`devfile.yaml`ファイルを選択します。
7. [編集] を選択します。

8. 開発ファイルを編集します。
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**
>
> 開発ファイルを編集する場合、変更を有効にするには開発ファイルを再起動する必要があります。これを実行すると実行できます。`/aws/mde/mde start --location devfile.yaml`devfile の起動に問題があると、リカバリモードに入ります。ただし、VPC に接続された開発環境に関連付けられている 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。

実行することで、どの devfile が使用されているかを確認できます。`/aws/mde/mde status`location フィールドには、環境のフォルダーを基準とした開発ファイルのパスが表示されます。`/projects`

```
{
    "status": "STABLE",
    "location": "devfile.yaml"
}
```

IDE の開発環境のリポジトリ開発ファイルを編集します。

開発環境の設定を変更するには、開発ファイル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、Dev Environment 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、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プルリクエストを作成する必要があります。

- の開発環境のリポジトリ開発ファイルを編集する AWS Cloud9
- VS Code での開発環境のリポジトリ開発ファイルの編集
- での開発環境のリポジトリ開発ファイルの編集 JetBrains

開発環境のリポジトリ開発ファイルの移動

デフォルト 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`

リカバリーモード

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"
}
```

下の口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 の機能は以下によってサポートされています。 CodeCatalyst

CodeCatalyst バージョン 2.0.0 では、次の devfile 機能がサポートされます。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`	コンポーネントプロパティ
`env`	コンポーネントプロパティ
`mountSources`	コンポーネントプロパティ
`volumeMounts`	コンポーネントプロパティ

例:開発環境に合わせた devfile の設定

以下は単純な 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 の起動ログ、コマンドログ、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。

開発ファイルコマンド

現在、devfile CodeCatalyst `exec` 内のコマンド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 exec で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"
```

Dev Environment に接続すると、定義したコマンドをターミナルから実行できます。

```
/aws/mde/mde command <command-id>
/aws/mde/mde command executescript
```

実行時間が長いコマンドの場合、-s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出力できます。

```
/aws/mde/mde -s command <command-id>
```

> **Note**
>
> command-id小文字でなければなりません。

がサポートする EXEC パラメーター CodeCatalyst

CodeCatalyst devfile バージョン exec 2.0.0 では以下のパラメーターをサポートします。

- commandLine
- component

- id

- workingDir

開発ファイルイベント

現在、devfile CodeCatalyst postStart 内のイベントのみをサポートしています。詳細については、postStartObjectDevFile.IO ドキュメントのを参照してください。

次の例は、devfile postStart に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、Dev Environment postStart は指定されたコマンドを定義された順序で実行します。コマンドが失敗した場合、Dev Environment は実行を継続し、実行出力は下のログに保存されます。/aws/mde/logs

開発ファイルコンポーネント

現在、devfile CodeCatalyst container 内のコンポーネント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 内のコンテナに起動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**
>
> コンテナに有効期間が短いエントリコマンドがある場合は、command: ['sleep', 'infinity']そのコマンドを追加してコンテナを稼働させ続ける必要があります。

CodeCatalyst また、コンテナコンポーネントの次のプロパティargs、、envmountSources、もサポートしていますvolumeMounts。

ユニバーサル開発ファイルイメージ

デフォルトのユニバーサルイメージには、IDE で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、 CodeCatalyst このイメージと、によって管理されるツールが含まれます CodeCatalyst。新しいイメージリリースの通知を受け取るには、を参照してくださいSNS によるユニバーサル画像通知。

> **Note**
>
> public.ecr.aws/aws-mde/universal-image:latestpublic.ecr.aws/aws-mde/universal-image:3.0イメージがイメージを取得します。

Amazon CodeCatalyst は以下の開発ファイルイメージをサポートしています。

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は、へのアップグレード後に PHP AWS Cloud9、Ruby、CSS `universal-image:3.0` でオートコンプリートが機能しなくなります。

トピック

SNS によるユニバーサル画像通知

CodeCatalyst ユニバーサル画像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アップデートがリリースされたときに通知する Amazon Simple Notification Service (SNS) トピックに登録できます。SNS トピックの詳細については、「Amazon 簡易通知サービスとは」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、購読者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アップデートを購読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 CodeCatalyst コンソールを使用してユニバーサルイメージアップデートを購読するには

1. Amazon SNS コンソールを開いて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」に、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] に E メールアドレスを入力します。この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-購読確認」という件名の確認メールが届きます。メールを開いて [購読を確認] を選択します。

Amazon SNS CodeCatalyst コンソールを使用してユニバーサルイメージアップデートの購読を解除するには

1. Amazon SNS コンソールを開いて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 [購読] を選択し、購読を解除したい購読を選択します。
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] をクリックします。

ユニバーサルイメージ 1.0 ランタイムバージョン

次の表は、universal-image:1.0で利用できるランタイムの一覧です。

universal-image:1.0ランタイムバージョン

ランタイム名	バージョン	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 コンポーズ	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	
php	7.2	`php: 7.x`
python	3.9	`python: 3.x`
	3.10	
ruby	3.1	`ruby: 3.x`
テラフォーム	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

次の表は、`universal-image:2.0`で利用できるランタイムの一覧です。

universal-image:2.0ランタイムバージョン

ランタイム名	バージョン	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 コンポーズ	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	
php	7.2	`php: 7.x`
python	3.9	`python: 3.x`
	3.10	
ruby	3.2	`ruby: 3.x`
テラフォーム	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

次の表は、`universal-image:3.0`で利用できるランタイムの一覧です。

universal-image:3.0ランタイムバージョン

ランタイム名	バージョン	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 コンポーズ	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`
php	8.2	`php: 8.x`
python	3.9	`python: 3.x`
	3.11	
ruby	3.2	`ruby: 3.x`
テラフォーム	1.5	`terraform: 1.x`

VPC 接続での開発環境の使用

VPC 接続は、VPC CodeCatalyst にアクセスするためのワークフローに必要なすべての設定を含むリソースです。スペース管理者は、スペースメンバーに代わって Amazon CodeCatalyst コンソールで独自の 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

クションを実行したり、ネットワークルールに準拠した開発環境を作成したり、関連する VPC 内のリソースにアクセスしたりできます。

開発環境を VPC 接続に関連付けることができるのは、開発環境の作成時のみです。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 Dev Environment を削除して、新しい Dev Environment を作成する必要があります。

> **Important**
>
> VPC 接続のある開発環境は、リンクされているサードパーティのソースリポジトリをサポートしていません。 CodeCatalyst

AWS 開発環境は作成時に複数のリソースとサービスを利用することに注意してください。つまり、開発環境は次のサービスに接続することになります。 AWS

- Amazon CodeCatalyst
- AWS SSM
- AWS KMS
- Amazon ECR
- Amazon CloudWatch
- Amazon ECS

> **Note**
>
> AWS Toolkit 関連する VPC 接続による開発環境の作成はサポートされていません。また AWS Cloud9、以外の IDE を使用する場合、読み込みに約 5 分かかる場合があることにも注意してください。

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、『CodeCatalyst 管理者ガイド』の「Amazon VPC CodeCatalyst の管理」を参照してください。

IDE での開発環境の使用

Dev Environments 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。Dev Environments は、サポートされている統合開発環境 (IDE) を備えた、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 からの操作について詳しくは、次のドキュメントを参照してください。

- CodeCatalyst JetBrains IDE 向けAmazon
- VS CodeCatalyst コード向けAmazon
- Amazon CodeCatalyst 用 AWS Cloud9

の開発環境のクォータ CodeCatalyst

次の表は、Amazon の開発環境のクォータと制限を示しています。 CodeCatalystAmazon 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、スペースの全体的な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のストレージ容量	開発環境のストレージは、スペースの全体的な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、スペースの全体的な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。

内のパッケージ CodeCatalyst

Amazon CodeCatalyst には、開発チームが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して共有することを容易にする完全マネージド型のパッケージリポジトリサービスがあります。これらのパッケージは、のプロジェクト内で作成、整理されたパッケージリポジトリに保存されます。 CodeCatalyst

CodeCatalyst 次のパッケージ形式をサポートします。

- npm

パッケージリポジトリ内のパッケージは、そのリポジトリを含むプロジェクトのメンバー間で検索および共有できます。

パッケージをリポジトリに追加するには、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公開したり、 CodeCatalyst パッケージリポジトリからパッケージを利用したりするワークフローを設定できます。 CodeCatalyst ワークフローでのパッケージの使用について詳しくは、を参照してくださいパッケージを操作する。

あるパッケージリポジトリ内のパッケージを上流リポジトリとして追加することで、同じプロジェクト内の別のリポジトリで利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。

CodeCatalyst 公開されている外部リポジトリをリポジトリに接続することで、オープンソースパッケージをリポジトリで利用できるようにすることができます。サポートされているリポジトリのリストなど、上流のリポジトリと外部リポジトリへの接続についての詳細は、を参照してください。アップストリームリポジトリを操作する

トピック

- パッケージの概念
- パッケージリポジトリの操作
- アップストリームリポジトリを操作する
- 公開されている外部リポジトリへの接続

- パッケージを操作する
- npmを使う
- パッケージのクォータ

パッケージの概念

ここでは、でパッケージを管理、公開、消費する際に知っておきたい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、ソフトウェアをインストールして依存関係を解決するの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。 CodeCatalyst npm パッケージ形式をサポートします。

パッケージは以下から構成されます。

- 名前 (例えば、webpackはよく使われる npm パッケージの名前)
- オプションの名前空間 (例:in) @types @types/node
- バージョンセット (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)

Package 名前空間

パッケージ形式によっては、パッケージを論理的なグループに整理して名前の衝突を避けるために、階層的なパッケージ名をサポートしているものもあります。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。たとえば、npm はスコープをサポートしており、npm @types/node パッケージのスコープはで、名前はです。@types 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 (「types」) はパッケージ名前空間と呼ばれ、名前 (「node」) 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と、名前の衝突を避けるのが難しくなる可能性があります。

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、.tgzパッケージバージョンに関連付けられて格納される個別のファイルです。 CodeCatalyst

Package 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パッケージセットが含まれ、それぞれがアセットセットに対応します。各パッケージリポジトリには、Node.js CLI () npm などのツールを使用してパッケージを取得して公開するためのエンドポイントがあります。1 つのスペースに最大 1,000 個のパッケージリポジトリを作成できます。

上流の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上流リポジトリとしてリンクすると、設定したリポジトリを通じて、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、「上流のリポジトリー」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認証局からパッケージを取得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の公式パッケージ認証局に接続された特別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オーソリティ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、代わりにすべてが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け取っ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、使用するプロジェクトごとに作成する必要があります。以下のリストには、 CodeCatalyst 作成可能なすべてのゲートウェイリポジトリと、それらが接続されているパッケージオーソリティが含まれています。

- npm-public-registry-gatewaynpmjs.com の npm パッケージを提供します。

上流のリポジトリー

CodeCatalyst を使用して、2 つのパッケージリポジトリ間の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に含まれるパッケージバージョンに、下流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、パッケージリポジトリは別のパッケージリポジトリの上流になります。上流の関係では、クライアント側から見ると、2 つのパッケージリポジトリの内容が効果的に統合されます。

たとえば、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要求した場合、 CodeCatalyst 設定されている上流のリポジトリでそのパッケージバージョンを検索します。上流の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を停止します。CodeCatalyst

パッケージリポジトリの操作

では CodeCatalyst、パッケージはパッケージリポジトリ内に保存され、管理されます。CodeCatalyst (CodeCatalyst またはサポートされている任意のパブリックパッケージリポジトリ) にパッケージを公開したり、パッケージを利用したりするには、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
- 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の編集
- 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の作成

以下の手順を実行して、にパッケージリポジトリを作成します CodeCatalyst。

パッケージリポジトリを作成するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. パッケージリポジトリを作成したい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 [Packages] を選択します。
4. 「Package 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. Package リポジトリの詳細セクションに、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、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. 「上流リポジトリの編集」セクションで、パッケージリポジトリを通じてアクセスしたいパッケージリポジトリを追加します。 CodeCatalyst Gateway リポジトリを追加して、外部のパッケージリポジトリや他のパッケージリポジトリに接続できます。 CodeCatalyst

   - パッケージリポジトリからパッケージがリクエストされると、上流のリポジトリはこのリストに表示されている順序で検索されます。パッケージが見つかると、 CodeCatalyst 検索を停止します。上流の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] ボタンを使用します。

7. [Create] 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に接続する。

パッケージを公開したり CodeCatalyst、 CodeCatalyst からパッケージを利用したりするには、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。リポジトリをまだ作成していない場合は、に記載されている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 CodeCatalyst 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、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上流リポジトリを編集するには、次の手順を実行します。

パッケージリポジトリを編集するには

1. https://codecatalyst.aws/ CodeCatalyst で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 [Packages] を選択します。
4. Package 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上流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を操作する」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を削除する。

でパッケージリポジトリを削除するには、次の手順を実行します CodeCatalyst。

パッケージリポジトリを削除するには

1. https://codecatalyst.aws/ CodeCatalyst で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 [Packages] を選択します。
4. Package 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、「削除」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ている情報を確認してください。
7. `delete`入力フィールドに入力し、[Delete] を選択します。

アップストリームリポジトリを操作する

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。上流のリポジトリを使用する主な利点は次のとおりです。

- パッケージマネージャーに 1 つのリポジトリエンドポイントを設定するだけで、複数のソースから取得できます。
- 上流のリポジトリから消費されたパッケージは下流のリポジトリに保存されるため、上流のリポジトリで予期せぬ停止が発生してもパッケージを利用できるようになります。

パッケージリポジトリを作成するときに、上流リポジトリを追加できます。コンソールで既存のパッケージリポジトリに対して上流リポジトリを追加または削除することもできます。 CodeCatalyst

ゲートウェイリポジトリを上流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上流リポジトリとしてリンクできます。たとえば、project-repoチームがという名前のリポジトリを作成し、npm-public-registry-gatewayそのリポジトリにが追加された別のリポジトリをすでに使用していて、team-repoそのリポジトリがパブリックの npm リポジトリに接続されているとします。npmjs.comteam-repoにアップストリームリポジトリとして追加できます。project-repoこの場合、、、project-repoteam-reponpm-public-registry-gateway、project-repoからのパッケージの取得に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

- 上流リポジトリの追加
- 上流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
- 上流のリポジトリを削除する

上流リポジトリの追加

CodeCatalyst パブリックパッケージリポジトリまたは別のパッケージリポジトリを上流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、ダウンストリームリポジトリに接続しているパッケージマネージャーが上流リポジトリ内のすべてのパッケージを利用できるようになります。

上流の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. Package 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したい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. 「上流リポジトリ」セクションで、「上流リポジトリを追加」を選択します。
5. 検索バーを選択すると、利用可能なリポジトリのリストが表示され、検索できます。CodeCatalyst サポートされているパブリックパッケージリポジトリやその他のリポジトリを上流リポジトリとして追加できます。追加したいリポジトリが見つかったら、リストから選択します。

> **Note**
>
> Gateway npm-public-registry-gatewayリポジトリにはリポジトリがあります。npmjs.comなどの公開されている外部パッケージ機関に接続するには、CodeCatalyst 外部リポジトリから引き出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ゲートウェイリポジトリを使用します。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、時間とデータ転送を節約できます。

6. アップストリームリポジトリとして追加したいリポジトリをすべて選択したら、[追加] を選択します。
7. 上流リポジトリの検索順序を変更する方法の詳細については、を参照してください。上流リポジトリの検索順序の編集

上流のリポジトリを追加すると、ローカルリポジトリに接続されたパッケージマネージャを使用して、上流の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。上流のリポジトリからパッケージバージョンをリクエストする方法の詳細については、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

上流リポジトリの検索順序の編集

CodeCatalyst 設定した検索順序で上流のリポジトリを検索します。パッケージが見つかると、CodeCatalyst 検索を停止します。上流のリポジトリでパッケージを検索する順序を変更できます。

上流の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、上流リポジトリの検索順序を編集したい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. 上流リポジトリセクションでは、上流リポジトリとその検索順序を確認できます。検索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] ボタンを使用します。
5. 上流のリポジトリの検索順序を編集し終えたら、[Save] 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

以下の例は、 CodeCatalyst パッケージマネージャーが上流の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リクエストした場合に考えられるシナリオを示しています。

この例では、npmなどのパッケージマネージャーが、downstream複数の上流リポジトリを含むという名前のパッケージリポジトリにパッケージバージョンを要求します。パッケージがリクエストされると、以下のことが起こ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。
- downstream要求されたパッケージバージョンが含まれていない場合は、 CodeCatalyst downstream設定されている検索順序でアップストリームのリポジトリで検索します。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いずれのアップストリームリポジトリにも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。 CodeCatalyst パッケージバージョンがリクエストされたときに検索されるリポジトリの最大数は 25 です。

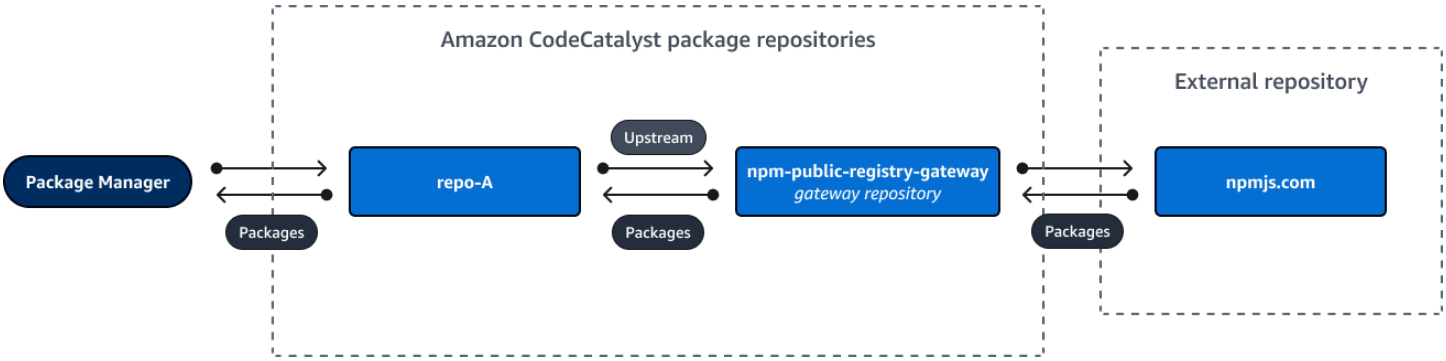
アップストリームリポジトリからのパッケージの保持

リクエストされたパッケージバージョンが上流の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いつでも利用できます。これにより、上流のリポジトリが予期せず停止した場合でも、パッケージにアクセスできるようになります。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) 。

上流のリレーションシップを通じてパッケージを取得する。

CodeCatalyst 上流リポジトリと呼ばれる複数のリンクされたリポジトリからパッケージを取得できる。 CodeCatalyst パッケージリポジトリが、 CodeCatalyst ゲートウェイリポジトリへの上流接続を持つ別のパッケージリポジトリへの上流接続を持っている場合、上流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、次のような構成を考えてみましょう。という名前のリポジトリは、repo-A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っています。npm-public-registry-gateway npm-public-registry-gateway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がある。



npmrepo-Aリポジトリを使用するように設定されている場合、running を実行すると https://npmjs.com npm install からへのパッケージのコピーが開始されます。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
```

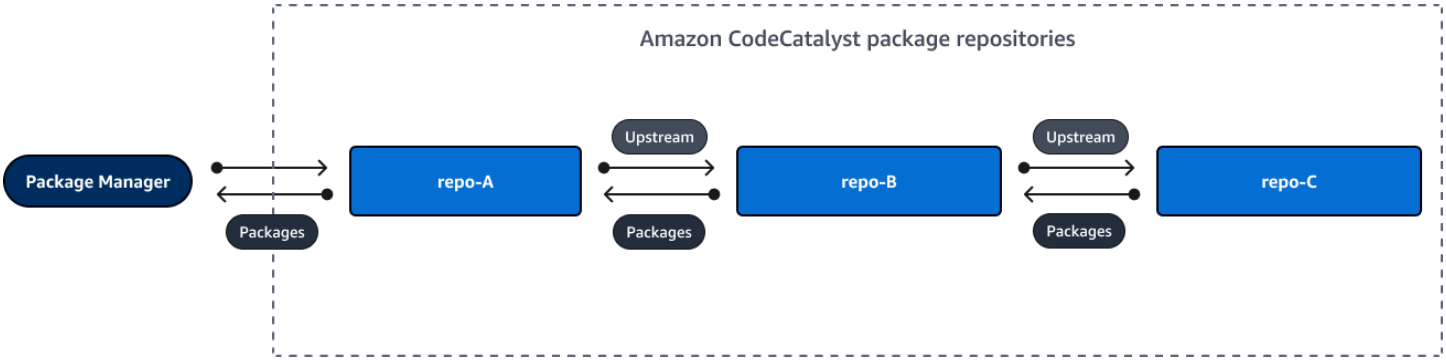
実行後はnpm install、npmから取得した最新バージョン (lodash 4.17.20) repo-A のみが含まれます。repo-A

npm-public-registry-gateway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パッケージバージョンはすべてに格納されます。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があれば、下流のリポジトリならどれでも取得できたはずです。npm-public-registry-gateway

npm-public-registry-gatewayの内容から、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時間の経過とともに表示できます。

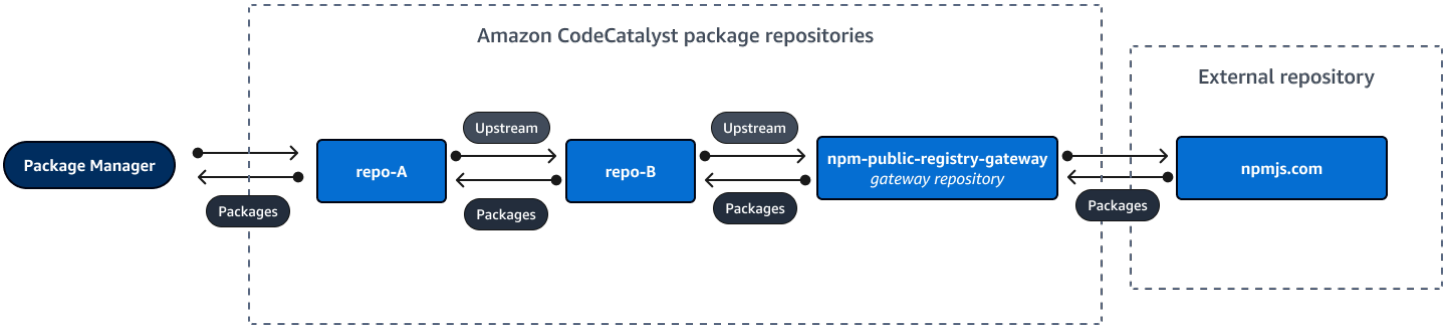
中間リポジトリでのパッケージの保持

CodeCatalyst 上流のリポジトリをチェーン化できます。たとえば、repo-Arepo-Bを上流のリポジトリとして持ち、repo-Brepo-C上流のリポジトリとして持つことができます。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。



repo-Aパッケージマネージャーがリポジトリに接続してリポジトリからパッケージバージョンを取得してもrepo-C、その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、一番遠い下流のリポジトリー (この例では) にのみ保持されます。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。たとえば、、、と 4 つのリポジトリがありrepo-Arepo-Brepo-Crepo-D、repo-A接続先のパッケージマネージャーがパッケージバージョンを取得した場合、パッケージバージョンはまたはには保持されますがrepo-D、またはには保持されません。repo-A repo-B repo-C

Package 保持の動作は、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取得する場合と似ていますが、パッケージバージョンがパブリックリポジトリに直接アップストリーム接続されているゲートウェイリポジトリに常に保持される点が異なります。たとえば、repo-Arepo-Bはアップストリームリポジトリとして機能します。 repo-Bnpm-public-registry-gatewayをアップストリームリポジトリとして使用し、パブリックリポジトリである npmjs.com へのアップストリーム接続を行います。下の図を参照してください。



接続先の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 (たとえば lodash 4.17.20) **repo-A** を要求し、そのパッケージバージョンが 3 つのリポジトリのどれにも存在しない場合は、npmjs.com

から取得されます。lodash 4.17.20 を取得すると、そのリポジトリは最も遠い下流のリポジトリであり、パブリックな外部リポジトリである **repo-A** npmjs.comへのアップストリーム接続があるため保持されます。**npm-public-registry-gateway** lodash 4.17.20 は中間リポジトリなので保持されません。 repo-B

上流のリポジトリを削除する

上流の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、上流の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
>
> 上流のリポジトリを削除すると、上流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壊れる可能性があります。

上流の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. 「Package リポジトリ」ページで、上流リポジトリを削除したい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. 「上流リポジトリ」セクションで、削除する上流リポジトリを探し、「削除」を選択します。
5. 上流のリポジトリを削除し終えたら、[保存] を選択します。

公開されている外部リポジトリへの接続

CodeCatalyst 対応するゲートウェイリポジトリを上流リポジトリとして追加することで、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の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取得し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、時間とデータ転送を節約できます。

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. ゲートウェイリポジトリのPackage リポジトリページでは、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、作成された日付と時刻が表示されます。作成されていない場合は、[Create] を選択して作成してください。
3. パブリックリポジトリに接続したいパッケージリポジトリを選択します。
4. 「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。
5. パブリックリポジトリに接続するには、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上流リポジトリとして追加します。

   「上流リポジトリの編集」セクションで、「リポジトリを追加」を選択します。 CodeCatalyst
6. 「ゲートウェイリポジトリ」セクションには、使用可能なゲートウェイリポジトリがすべて一覧表示されます。接続したい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が見つかったら、リストから選択して [追加] を選択します。
7. リポジトリからパッケージがリクエストされると、「Edit Upstream repositories」CodeCatalyst リストに表示されている順序で上流のリポジトリを検索します。パッケージが見つかると、検索を停止します。 CodeCatalyst 上流の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、並び替え矢印を使用します。
8. 上流リポジトリの追加と順序付けが完了したら、[Save] を選択します。

ゲートウェイリポジトリを上流リポジトリとして追加したら、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を使用して、それに対応する公開の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たパッケージは、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。上流のリポジトリからパッケージバージョンをリクエストする方法の詳細については、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 ゲートウェイリポジトリを含む以下の公式パッケージ認証局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`

パッケージを操作する

CodeCatalyst 内のパッケージは、依存関係を解決してソフトウェアをインストールするのに必要なソフトウェアとメタデータのバンドルです。 CodeCatalyst npm パッケージ形式をサポートします。このセクションでは、パッケージの公開、表示、削除、およびパッケージバージョンのステータスの更新に関する情報を提供します。

トピック

- メッセージの公開
- パッケージバージョンの詳細を表示する
- 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

メッセージの公開

パッケージマネージャーツールを使用して、 CodeCatalyst サポートされている任意の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公開できます。

CodeCatalyst 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てパッケージを公開する方法については、を参照してくださいパッケージリポジトリに接続する。。

目次

- 公開リポジトリとアップストリームリポジトリ
- プライベートパッケージと公開リポジトリ
- パッケージアセットの上書き

公開リポジトリとアップストリームリポジトリ

では CodeCatalyst、アクセス可能な上流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。たとえば、npm パッケージをパッケージリポジトリに公開したいが`lodash@1.0`、`myrepo`上流リポジトリとして設定されたゲートウェイリポジトリ経由で npmjs.com `myrepo` に接続されているとします。`lodash@1.0`が上流のリポジトリまたはnpmjs.comに存在する場合、409 CodeCatalyst コンフリクトエラーを発行して公開を試みても拒否します。`myrepo`これにより、予期しない動作を引き起こす可能性のある、上流のリポジトリにある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ことを防ぐことができます。

上流の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 CodeCatalyst リポジトリに保存されているパッケージを npmjs.com などのパブリックリポジトリに公開しません。 CodeCatalyst パブリックリポジトリからリポジトリにパッケージをインポートしますが、パッケージを逆方向に移動させることはありません。 CodeCatalyst CodeCatalyst リポジトリに公開したパッケージはプライベートのままで、 CodeCatalyst リポジトリが属する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

すでに存在していて、別のコンテンツが含まれ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、まずその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を表示する

CodeCatalyst コンソールを使用して、特定のパッケージバージョンに関する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. Package リポジトリページで、詳細を表示したい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ーできます。一覧からパッケージを選択します。

4. 「Package 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Package バージョン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. Package 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. 表からパッケージを検索して選択します。
4. Package 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. 「Package 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、「削除」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

CodeCatalyst のすべてのパッケージバージョンには、そのパッケージバージョンの現在の状態と可用性を示すステータスがあります。 CodeCatalystパッケージバージョンのステータスはコンソールで変更できます。パッケージバージョンに設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. Package リポジトリページで、ステータスを更新したい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. 表からパッケージを検索して選択します。
4. 「Package 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. 「Package 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「廃棄」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、更新するステータスに応じて [リスト解除]、[アーカイブ]、または [廃棄] 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ステータスに指定できる値は次のとおりです。パッケージバージョンステータスはコンソールで変更できます。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から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 (の出力など) `npm view <package-name> versions` に含まれます。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。
- 限定公開: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リストには含まれません。例えば、npm パッケージの場合、`npm view <package-name> versions`の出力にはパッケージバージョンは含まれません。つまり、npm の依存関係解決ロジックではパッケージのバージョンは選択されません。なぜなら、そのバージョンは利用可能なバージョンのリストに表示されないからです。ただし、**`npm package-lock.json`**リストに記載されていないパッケージバージョンが既にファイル内で参照されている場合でも、実行中などには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 Archived 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 Unlist または Published に戻すことはできます。
- 廃棄:パッケージバージョンはリストに表示されず、アセットはリポジトリからダウンロードできません。「廃棄済み」と「アーカイブ済み」の主な違いは、ステータスが「破棄」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る点です。 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。パッケージバージョンはアセットが削除されているため使用できません。パッケージバージョンが「破棄」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ついては請求されません。

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。削除すると、パッケージバージョンはリポジトリに存在しなくなるため、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、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したり、上流のリポジトリからプルダウンしたり、外部のパブリックリポジトリから取り込んだりすることで、パッケージリポジトリに追加できます。直接公開とパブリックリポジトリからの取り込みの両方でパッケージのバージョンを追加できるようにすると、依存性代替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。依存性代替攻撃から身を守るには、リポジトリ内のパッケージにパッケージオリジンコントロールを設定し、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が、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提供されるように、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してください。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る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:どのパッケージバージョンも、 CodeCatalyst アップストリーム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、外部接続を使用して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:Package バージョンは、 CodeCatalyst アップストリーム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、外部接続を使用して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をパッケージリポジトリに追加する方法に基づいています。

- 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

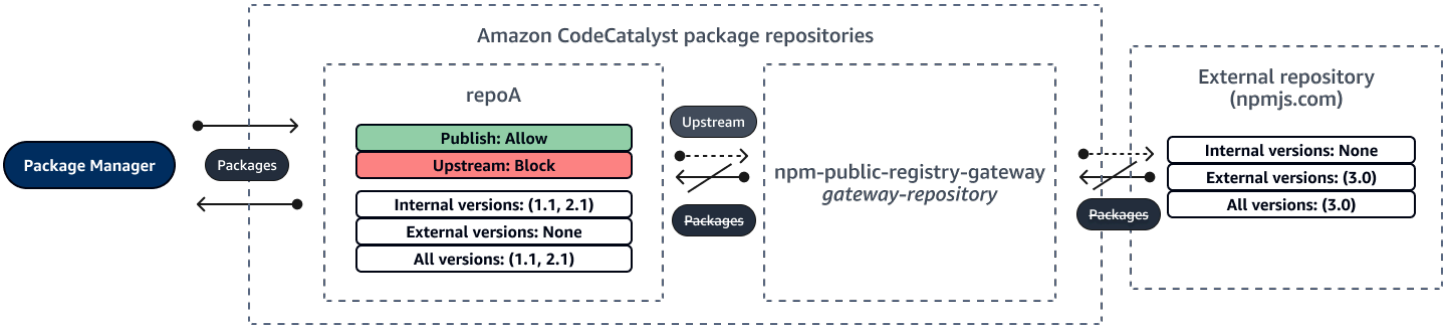
このセクションでは、 CodeCatalyst パッケージバージョンがパッケージリポジトリに追加される場合の一般的なシナリオをいくつか説明します。Package オリジンコントロール設定は、最初のパッケージバージョンの追加方法に応じて、新しいパッケージに設定されます。

以下のシナリオでは、管理しているパッケージなど、内部パッケージがパッケージマネージャーから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パッケージリポジトリに公開します。 CodeCatalyst 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、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。 CodeCatalyst これは、内部パッケージに対する依存性代替攻撃が試みられた場合もあれば、偶然の場合もあります。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。

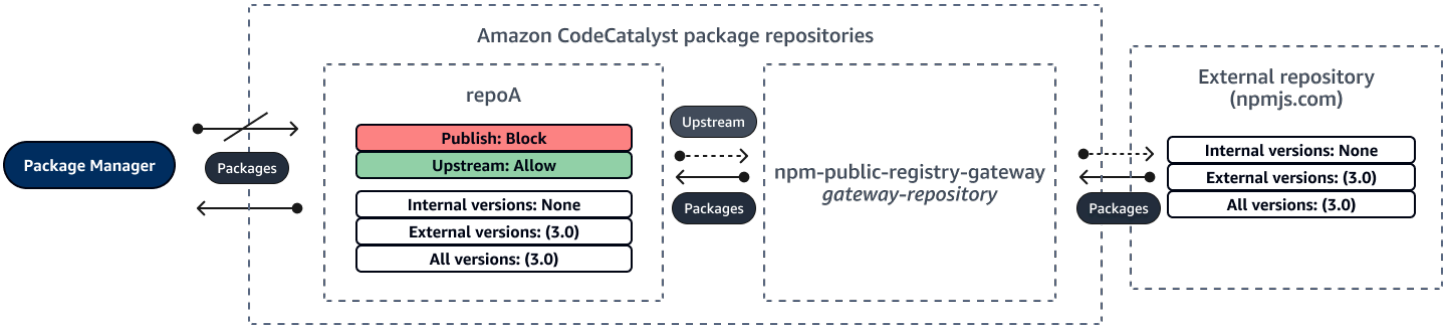
以下の画像では、RePoA CodeCatalyst はパブリックリポジトリへの外部接続を備えたパッケージリポジトリです。リポジトリには packageA のバージョン 1.1 と 2.1 が含まれていますが、バージョン 3.0 はパブリックリポジトリに公開されています。通常、RePoA はパッケージマネージャーからパッケージがリクエストされた後にバージョン 3.0 を取り込みます。パッケージ取り込みは Block に設定されているため、バージョン 3.0 CodeCatalyst はパッケージリポジトリに取り込まれず、接続しているパッケージマネージャーも使用できません。



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、PackageB というパッケージは、リポジトリに接続したパブリックリポジトリの外部に存在します。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。公開パッケージを知らず、関連性のないパッケージを同じ名前で公開しようとしている、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、あるいは外部にすでに存在するパッケージバージョンとまったく同じものを直接公開しようとしている可能性があります。 CodeCatalyst 公開しようとしているバージョンを拒否しますが、必要に応じて拒否を明示的に無効にし、バージョンを公開できます。

以下の画像では、RePoA CodeCatalyst はパブリックリポジトリへの外部接続を備えた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 に公開できますが、公開が Block に設定されているため、バージョン 1.2 は公開できません。



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、PackageB というパッケージが、パッケージリポジトリに接続しているパブリックリポジトリに外部に存在します。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が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

Package 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る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、以下の手順のステップを実行します。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

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. パッケージ概要ページから、「オリジンコントロールを編集」を選択します。

5. 「Origin Controls」で、このパッケージに設定したいパッケージオリジンコントロールを選択します。「パブリッシュ」と「アップストリーム」の両方のパッケージオリジンコントロール設定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、アクセス可能な上流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。たとえば、npm パッケージをリポジトリに公開したいがmyrepo、上流のリポジトリに npmjs.com `lodash@1.0` `myrepo` への外部接続があるとします。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`が上流のリポジトリまたは npmjs.comに存在する場合、409 CodeCatalyst コンフリクトエラーを発行して公開を試みても拒否します。`myrepo`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はアクセスできないため、`lodash`まだ存在していないどのバージョンのでもリポジトリに公開できます。
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。

依存性代替攻撃は、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が「混乱」または「置換」され、コードが危険にさらされます。

依存性代替攻撃を防ぐために、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。この統制は、新しいパッケージの最初のパッケージバージョンがリポジトリに追加されたときに自動的に設定されます。この統制により、 CodeCatalyst パッケージバージョンをリポジトリに直接公開したり、公開ソースから取り込んだりすることができないため、依存性代替攻撃からユーザーを保護できます。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

以下のトピックではnpm、Node.js パッケージマネージャーの使い方について説明しています。CodeCatalyst

> **Note**
>
> CodeCatalyst node v4.9.1とそれ以降、npm v5.0.0およびそれ以降のサポート。

トピック

- npm の設定と使用
- npm タグ処理

npm の設定と使用

を使用するにはnpm CodeCatalyst、npmパッケージリポジトリに接続し、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。npm CodeCatalyst パッケージリポジトリへの接続手順はコンソールで確認できます。

目次

npm を次のように設定します。 CodeCatalyst

以下の手順では、npm CodeCatalyst パッケージリポジトリを認証して接続する方法について説明します。npm について詳しくは、npm の公式ドキュメントを参照してください。

npmパッケージリポジトリに接続するには CodeCatalyst

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、[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Connect] を選択します。
6. 「構成の詳細」の「Package マネージャークライアント」で、「npm client」を選択します。
7. オペレーティングシステムを選択すると、対応する設定手順が表示されます。
8. で npm を認証するには個人アクセストークン (PAT) が必要です。 CodeCatalystトークンを既にお持ちの場合は、それを使用できます。持っていない場合は、以下の手順で作成できます。

   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管します。

> **Warning**
>
> ダイアログボックスを閉じると、PAT を再度表示したりコピーしたりできなくなります。認証情報は、攻撃者が不正に流用した後にその認証情報を使用できる時間を最小限に抑えるため、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以下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。コマンドは以下のことを行います。

- `.npmrc`プロジェクトにプロジェクトレベルのファイルがない場合は、プロジェクトレベルのファイルを作成します。
- `.npmrc`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ファイルに追加します。
- 認証情報 (PAT) をユーザーレベルのファイルに追加します。`.npmrc`

次の値を置き換えてください。

> **Note**
>
> コンソールの指示からコピーする場合、以下のコマンドの値は自動的に更新されるので、変更する必要はありません。

- *username CodeCatalyst ##################*。
- *PAT #### PAT* に置き換えてください。 CodeCatalyst
- *space_name ##################*。 CodeCatalyst
- *proj_name* を自分のプロジェクト名に置き換えてください。 CodeCatalyst
- *repo_name ##########################*。 CodeCatalyst

```
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:GETリクエストがあっても npm が必ず認証トークンをに渡すようにするには CodeCatalyst、always-auth 設定変数を次のように設定します。npm config set

```
npm set //packages.region.codecatalyst.aws/npm/space-name/proj-name/repo-
name/:always-auth=true --location project
```

npm パッケージをパッケージリポジトリからインストールする CodeCatalyst

の手順に従って npm をリポジトリに接続するとnpm を次のように設定します。 CodeCatalyst、npmリポジトリ上でコマンドを実行できます。

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にある npm CodeCatalyst パッケージをコマンドでインストールできます。npm install

```
npm install lodash
```

npmjs から npm パッケージをインストールするには CodeCatalyst

CodeCatalyst npmjs.comに接続されたゲートウェイリポジトリへのアップストリーム接続でリポジトリを設定することで、npmjs.comからリポジトリ経由でnpmパッケージをインストールできます。npm-public-registry-gatewaynpmjs 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のパッケージリポジトリに取り込まれ、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. まだ行っていない場合は、npm CodeCatalystの手順に従ってパッケージリポジトリで設定してください。npm を次のように設定します。 CodeCatalyst
2. リポジトリがアップストリーム接続としてゲートウェイリポジトリを追加していることを確認します。npm-public-registry-gatewayの指示に従ってリポジトリを選択することで、npm-public-registry-gatewayどのアップストリームソースがアップストリームソースとして追加されたか、上流リポジトリの追加またはアップストリームソースとして追加されているかを確認できます。npm-public-registry-gateway
3. 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

```
npm install package_name
```

上流のリポジトリからパッケージをリクエストする方法の詳細については、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

npm パッケージをパッケージリポジトリに公開する CodeCatalyst

完了するとnpm を次のように設定します。 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 CodeCatalyst パッケージをパッケージリポジトリに公開できます。

```
npm publish
```

npm パッケージの作成方法については、「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、npm CodeCatalystパッケージリポジトリでサポートされているコマンドと、サポートされていない特定のコマンドを一覧表示しています。

トピック

- パッケージリポジトリとやりとりするサポート対象のコマンド
- サポートされているクライアント側コマンド
- サポートされていないコマンド

パッケージリポジトリとやりとりするサポート対象のコマンド

このセクションには、npmクライアントが構成先のレジストリー (例:npm config set registry) に 1 npm つまたは複数の要求を行うコマンドが一覧表示されます。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確認されています。

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。

コマンド	説明
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、`--browser` config パラメーターを使用して開こうとします。
doctor (ドクター)	一連のチェックを実行して、npm JavaScript のインストール環境がパッケージを管理できるかどうかを検証します。
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`、その直後にを実行します。`npm test`
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`、その直後にを実行します`npm test`。
outdated (旧式)	設定し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ないかを判断します。
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、`--browser` config パラメーターを使用して開こうとします。

コマンド	説明
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

CodeCatalyst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、サポートするものは何も必要ありません。

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。
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)	ローカルのパッケージツリーを検索し、依存関係をツリーのさらに上に移動することで構造を簡略化しようとします。これにより、依存パッケージを複数の依存パッケージでより効果的に共有できるようになります。
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から依存関係を選択し、そのパッケージディレクトリをデフォルトエディタで開きます。
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。指定し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生成します。コマ

コマンド	説明
	ンドを指定すると、そのコマンドはサブシェルで実行され、サブシェルはただちにシャットダウンします。
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。特に指定されていない限り、`package.json -g`これはファイルを格納する最も近い親ディレクトリです。
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、および関連するプレスクリプトとポストスクリプトを実行します。
root (ルート)	`node_modules` 有効な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。

コマンド	説明
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

npm CodeCatalyst これらのコマンドは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 公開されている npmjs リポジトリとは異なる権限モデルを使用しています。
adduser	レジストリユーザーアカウントを追加します。	CodeCatalyst 公開されている npmjs リポジトリとは異なるユーザーモデルを使用する。
audit (監査)	セキュリティ監査を実行します。	CodeCatalyst 現在、セキュリティ脆弱性データは販売していません。
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 現在、変更通知メカニズムはサポートされていません。
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 公開されている npmjs リポジトリとは異なる認証モデルを使用する。詳細については、npm を次のように設定します。 CodeCatalyst を参照してください。

コマンド	説明	メモ
logout (サインアウト)	レジストリからサインアウトします。	CodeCatalyst 公開されている npmjs リポジトリとは異なる認証モデルを使用する。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が過ぎると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 公開されている npmjs リポジトリとは異なる権限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 公開されている npmjs リポジトリとは異なるユーザーモデルを使用する。
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 コマンドはサポートされていません。`search`
star (星)	お気に入りのパッケージをマークします。	CodeCatalyst 現在、お気に入りメカニズムはサポートされていません。
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 現在、お気に入りメカニズムはサポートされていません。
team (チーム)	チームとチームメンバーシップを管理します。	CodeCatalyst 公開されている npmjs リポジトリとは異なるユーザーとグループのメンバーシップモデルを使用します。

コマンド	説明	メモ
token (トークン)	認証トークンを管理します。	CodeCatalyst 認証トークンの取得には別のモデルを使用します。詳細については、npm を次のように設定します。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 npm クライアントを使用してリポジトリからパッケージバージョンを削除することはサポートされていません。パッケージはコンソールで削除できます。
whoami (私は誰)	npm ユーザー名を表示します。	CodeCatalyst 公開されている 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。バージョン番号の代わりにタグを使用してエイリアスを指定できます。たとえば、複数の開発ストリームを含むプロジェクトがあり、ストリームごとに異なるタグ (、、stablebetadev、などcanary) を使用しているとします。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版リリースバージョンの最新タグのみを使用します。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 クライアントでタグを編集する

3 npm dist-tag つのコマンド (add、rm、およびls) は、デフォルトの npm CodeCatalyst レジストリで機能するのと同じようにパッケージリポジトリでも機能します。

npm タグと上流リポジトリ

npmリクエストにより、パッケージのタグとそのパッケージのバージョンが上流のリポジトリにも存在する場合、 CodeCatalyst クライアントに返す前にタグをマージします。たとえば、Rという名前のリポジトリには、という上流のリポジトリがあります。U次の表は、web-helper両方のリポジトリに存在する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 *web-helper* クライアントがリポジトリからパッケージのタグを取得すると*R*、latest タグと alpha 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ある場合は、CodeCatalyst 最後に更新されたタグを使用します。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 web-helper パッケージのタグを取得すると、最新のタグは最後に更新されたため*R*、バージョン 1.0.1 のエイリアスを付けます。これにより、ローカルリポジトリにまだ存在していない上流のリポジトリにある新しいパッケージバージョンを実行することで簡単に利用できるようになります。npm update

パッケージのクォータ

次の表は、Amazon のパッケージのクォータと制限について説明しています。 CodeCatalystAmazon 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
Package リポジトリ	1 スペースあたり最大 1000 個。
ダイレクト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。
上流の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最大 25 の上流リポジトリが検索されます。
Package 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 5 GB。
Package アセット	パッケージバージョンあたり最大 150 個。

のワークフローによるビルド、テスト、デプロイ CodeCatalyst

CodeCatalystDev Environment CodeCatalyst でアプリケーションコードを記述して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、ワークフローを使用することです。

ワークフローは、継続的インテグレーションと継続的デリバリー (CI/CD) システムの一部としてコードをビルド、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、つまりアクションを定義します。ワークフローでは、ワークフローを開始させるイベント、つまりトリガーも定義されます。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エディターまたは YAML エディター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、ブループリントを使用してプロジェクトを作成してください。各ブループリントには、レビュー、実行、実験が可能な、機能するワークフローが導入されています。

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、ワークフローを記述する YAML ファイルです。このファイルは、~/.codecatalyst/workflows/ソースリポジトリのルートにあるフォルダーに保存されます。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。

以下は簡単なワークフロー定義ファイルの例です。次の表では、この例の各行について説明します。

```
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トップレベルのプロパティ。
`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トップレベルのプロパティ。
`RunMode: QUEUED`	CodeCatalyst 複数の実行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キュー実行、代替実行、parallel実行の設定。
`Triggers:`	ワークフローの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、「トリガーの使用」を参照してください。

`- Type: PUSH`
`  Branches:`
`    - main`
	mainデフォルトソースリポジトリのブランチにコードをプッシュするたびにワークフローが開始され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ソースの操作。

`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、ActionsBuildセクションではという単一のアクションを定義していま

線グラフ	説明
	す。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の使用。
`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でのワークフローを使用した構築 CodeCatalyst。
`Identifier: aws/build@v1`	ビルドアクションのハードコードされた一意の識別子を指定します。

`Inputs:`
`  Sources:`
`  - WorkflowSource`
	`WorkflowSource` ビルドアクションがソースリポジトリを検索して、処理を完了するの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、「ソースの操作」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。

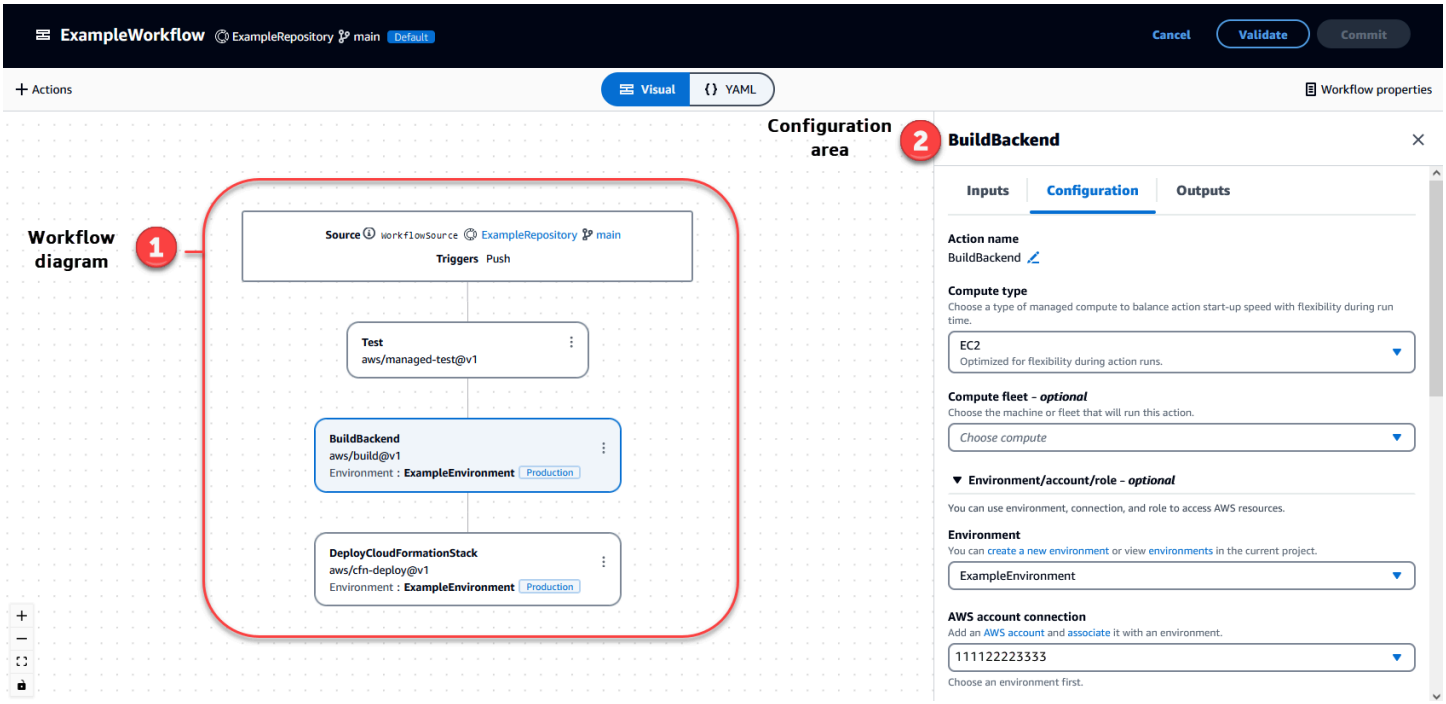
`Steps:`
`  - Run: docker build -t MyApp:lat`
`est .`
	という名前の Docker MyApp イメージを構築し、タグを付けるようにビルドアクションに指示します。`latest`

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一覧については、ワークフロー定義リファレンスを参照してください。

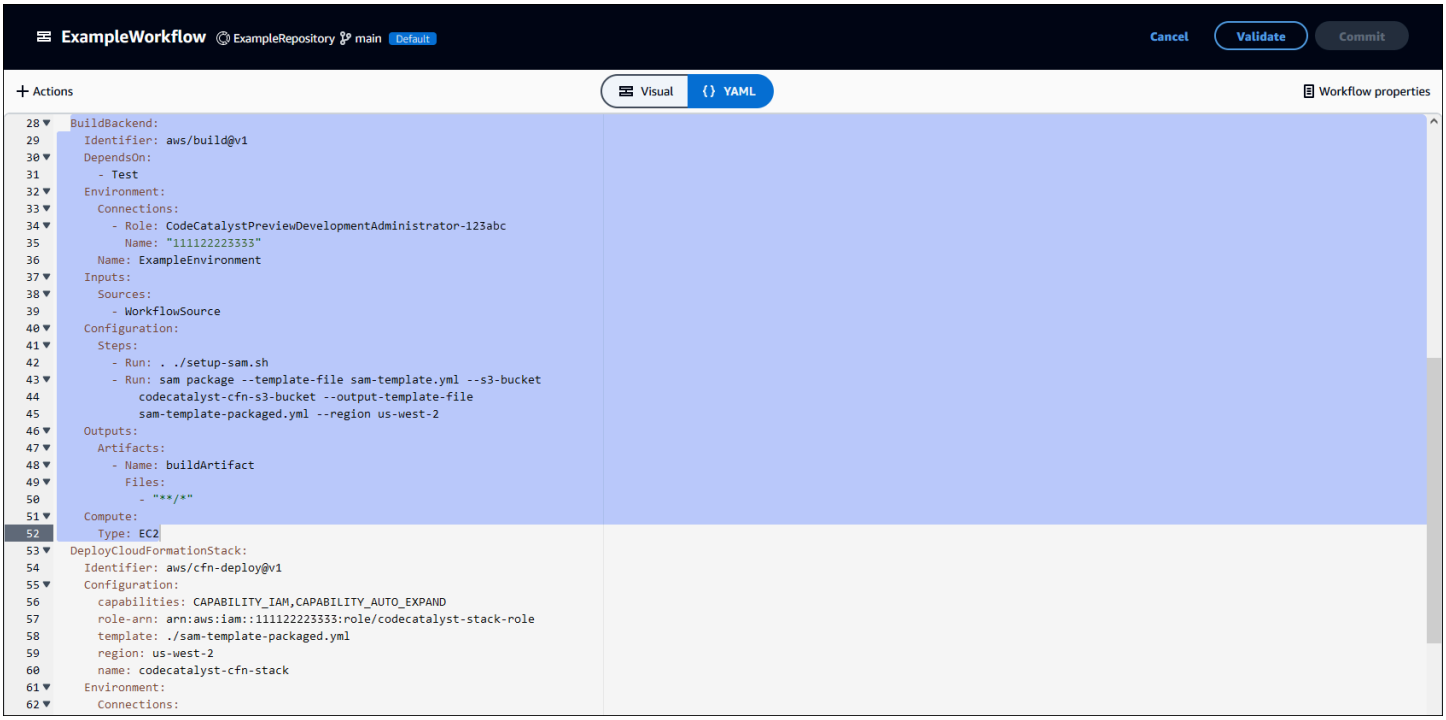
CodeCatalyst コンソールのビジュアルエディタと YAML エディタの使用

ワークフロー定義ファイルの作成と編集には好みのエディターを使用できますが、 CodeCatalyst コンソールのビジュアルエディターまたは YAML エディターの使用をお勧めします。これらのエディタは、YAML プロパティ名、値、入れ子、間隔、大文字小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に役立ちます。

以下の画像は、ビジュアルエディターのワークフローを示しています。ビジュアルエディターには、ワークフロー定義ファイルを作成して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ーには、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) があります。



別の方法として、次の画像に示す YAML エディターを使用することもできます。YAML エディターを使用して (チュートリアルなどの) 大きなコードブロックを貼り付けたり、ビジュアルエディターでは提供されていない高度なプロパティを追加したりできます。

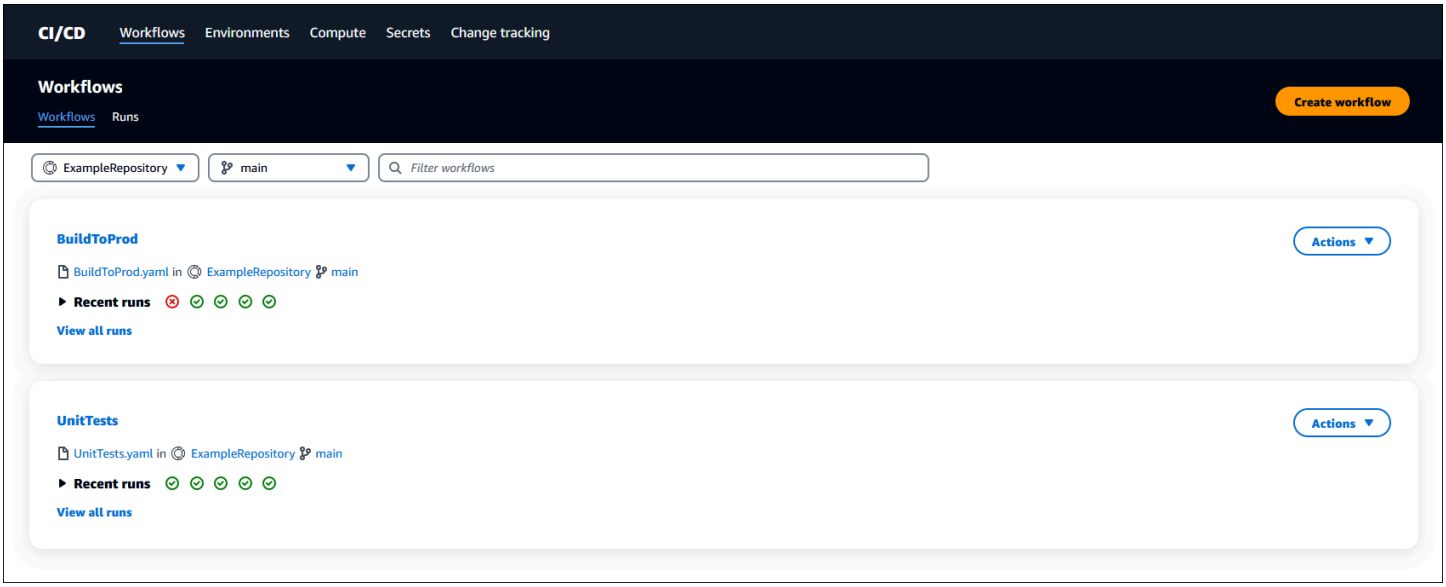


ビジュアルエディターから YAML エディターに切り替えて、構成が基盤となる YAML コードに与える影響を確認できます。

ワークフローを発見する

ワークフローは、同じプロジェクトで設定した他のワークフローとともに、ワークフローの概要ページで確認できます。

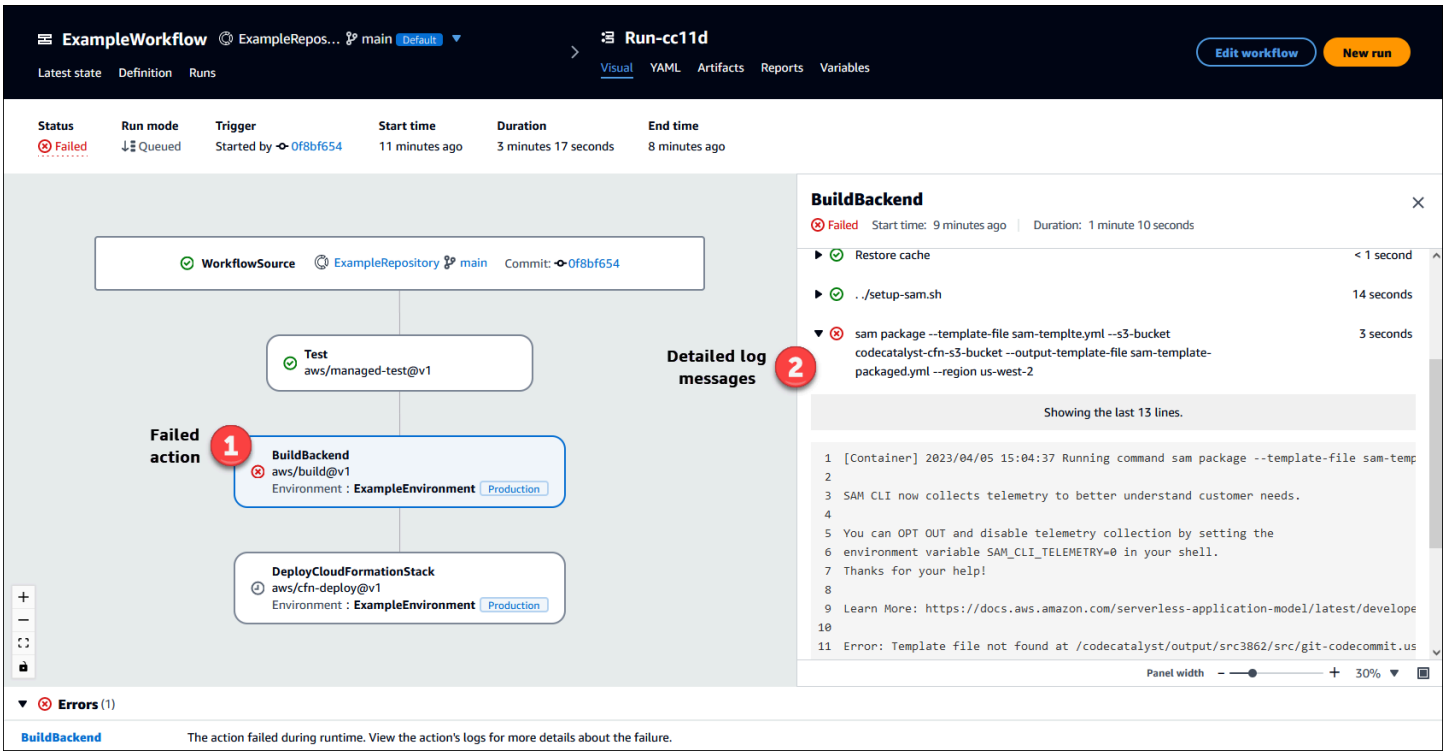
以下の画像は、ワークフローの概要ページを示しています。このページには、と 2 つのワークフローが入力されています。BuildToProdUnitTests両方とも数回実行されたことがわかります。[最近の実行] を選択して実行履歴をすばやく確認したり、ワークフローの名前を選択してワークフローの YAML コードやその他の詳細情報を確認したりできます。



ワークフロー実行の詳細を表示する

ワークフローの概要ページで実行を選択すると、実行されたワークフローの詳細を表示できます。

以下の画像は、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) に移動すると、詳細なログメッセージを表示したり、問題をトラブルシューティングしたりできます。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ランの処理。



次のステップ

ワークフローの概念について詳しくは、を参照してくださいワークフローの概念。

初めてのワークフローを作成するには、を参照してくださいでのワークフロー入門 CodeCatalyst。

ワークフローの概念

ワークフローを使用してコードをビルド、テスト、またはデプロイする際に知っておく必要がある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と継続的デリバリー (CI/CD) システムの一部としてコードをビルド、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、つまりアクションを定義します。ワークフローでは、ワークフローを開始させるイベント、つまりトリガーも定義されます。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エディターまたは YAML エディター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、ブループリントを使用してプロジェクトを作成してください。各ブループリントには、レビュー、実行、実験が可能な、機能するワークフローが導入されています。

ワークフローの詳細については、「ワークフローでの作業」を参照してください。

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。このファイルは、`~/.codecatalyst/workflows/`ソースリポジトリのルートにあるフォルダーに保存されます。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。ワークフロー定義リファレンス

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論理的な作業単位を定義します。通常、ワークフローには、設定方法に応じて順次またはparallel 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アクションの使用。

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、異なるアクション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。

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、通常はフォルダまたはファイルのアーカイブで構成されます。アーティファクトはアクション間でファイルや情報を共有できるので重要です。

アーティファクトの詳細については、「アーティファクトによる作業」を参照してください。

コンピューティング

コンピュートとは、 CodeCatalyst ワークフローアクションを実行するために管理・保守され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。

コンピュートについて詳しくは、を参照してくださいコンピュートでの作業。

環境

開発環境と混同しないように、環境はコードのデプロイ先です。通常、環境には実行中のアプリケーションのインスタンスとそれに関連するインフラストラクチャーが含まれます。環境には、開発、テスト、ステージング、プロダクションなどの名前を付けることができます。 CodeCatalyst によって環境に対して生成されたデプロイメントはすべて、Environments ページに表示されます。環境を設定するには、などの名前を付けてからmy-production-environment、と関連付けます。 AWS アカウント

環境は、デプロイ情報を表示するだけでなく、 AWS IAM ロールをワークフローアクションに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する。

レポート

レポートには、ワークフローの実行中に行われるテストに関する詳細が含まれます。テストレポート、コードカバレッジレポート、ソフトウェアコンポジション分析レポート、スタティック解析レポートなどのレポートを作成できます。レポートは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失敗率を確認できるため、アプリケーションとデプロイ設定を最適化するのに役立ちます。

レポートの詳細については、「テストレポートタイプ」を参照してください。

実行

実行とは、ワークフローを 1 回繰り返すことです。実行中、 CodeCatalystワークフロー設定ファイルに定義されている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ランの処理。

[Sources] (出典)

ソースは入力ソースとも呼ばれ、ワークフローアクションがタスクを実行するためにアクセスする必要のあるソースリポジトリです。たとえば、ワークフローアクションは、単体テストを取得してアプリケーションのソースファイルに対して実行するために、ソースにアクセスする必要がある場合があります。

sources の詳細については、「ソースの操作」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。変数の操作

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自動的に実行されるワークフローを開始できます。 CodeCatalyst ソフトウェア開発

者がコンソールからワークフローを手動で開始しなくても済むように、トリガーを設定するとよいでしょう。

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は、コミットがプッシュされるたびにワークフローの実行を開始します。
- 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、修正、またはクローズされるたびにワークフローの実行を開始します。
- スケジュール — スケジュールのトリガーにより、定義したスケジュールでワークフローの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することを検討してください。そうすれば、ソフトウェア開発者は翌朝に最新のビルドに取り組むことができます。

プッシュ、プルリクエスト、スケジュールの各トリガーは、単独で使用することも、同じワークフロー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、「トリガーの使用」を参照してください。

でのワークフロー入門 CodeCatalyst

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を学習します。

> **Tip**
>
> 事前に設定されたワークフローから始めるのが好きですか? ワークフローブループリントを使ったプロジェクトの作成、サンプルアプリケーション、その他のリソースが機能するようにプロジェクトを設定する方法については、を参照してください。

トピック

- （オプション）ステップ 4: クリーンアップする

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、「AWS Builder ID ユーザーをサポートするスペースの作成」を参照してください。
- CodeCatalyst スペースには、次の名前の「ゼロから開始」 CodeCatalyst という空のプロジェクトが必要です。

  ```
  codecatalyst-project
  ```

  詳細については、「Amazon で空のプロジェクトを作成する CodeCatalyst」を参照してください。
- プロジェクトには、 CodeCatalyst 以下の名前の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れば使用できますが、新しいプロジェクトとソースリポジトリを作成すると、このチュートリアルの最後で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、変更が加えられたときにソースコードを自動的にビルド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。

CodeCatalyst ワークフロー定義ファイルがコンソールの YAML エディターに表示されます。

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ーまたは YAML エディターで設定できます。YAML エディターから始めて、ビジュアルエディターに切り替えましょう。

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. Build アクションで + を選択し、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。これで、ワークフローは次のようになりました。

```
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、WorkflowSourceBuild_f0ソースリポジトリ内のファイルをアクションを実行しているコンピュートマシンにコピーし、Hello, World!ログに出力し、コンピュートマシン上の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. Visual を選択すると、ワークフロー定義ファイルがビジュアルエディターに表示されます。ビジュアルエディターのフィールドでは、YAML エディターに表示される YAML プロパティを設定できます。

ステップ 2: ワークフローをコミットして保存する

このステップでは、変更を保存します。.yamlワークフローはリポジトリにファイルとして保存されるため、変更をコミットとともに保存します。

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、ワークフロー設定ファイルの名前 (など**my-first-workflow**) を入力します。
4. 「コミットメッセージ」に、コミットを識別するメッセージ (など) を入力します**create my-first-workflow.yaml**。
5. 「リポジトリ」で、ワークフローを保存するリポジトリ (codecatalyst-repository) を選択します。
6. [ブランチ名] で、ワークフローを保存するブランチ (main) を選択します。
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。表示されるまでにはしばらく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が自動的に実行されます。

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 (例:) を選択しますWorkflow_fe47。

   ソースリポジトリ (WorkflowSource) のラベル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]、[構成]、[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を表示する」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:

1. 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、そのプロジェクトを削除してください。手順については、「Amazon でプロジェクトを削除する CodeCatalyst」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。
2. ワークフローがまだ削除されていない場合は削除してください。手順については、「ワークフローを削除するには」を参照してください。
3. リポジトリがまだ削除されていない場合は削除します。手順については、ソースリポジトリを削除する を参照してください。

コミットインごとのコード品質とデプロイステータスの表示 CodeCatalyst

開発ライフサイクルのどの時点でも、バグ修正、新機能、その他の影響の大きい変更など、特定のコミットのデプロイ状況を把握することが重要です。デプロイ状況追跡機能が開発チームに役立つ次のようなシナリオを考えてみましょう。

- 開発者は、バグに対処するために修正を行ったので、チームのデプロイ環境全体にわたるそのリリース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は、デプロイされたコミットのリストを見て、そのデプロイ状況を追跡して報告したいと思うでしょう。

CodeCatalyst 個々のコミットや変更がどこにデプロイされ、どの環境にデプロイされたかを一目で判断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メントのステータス。
- コミットが正常にデプロイされた環境。
- CI/CD ワークフロー内のコミットに対して実行されたテストのステータス。

以下の手順では、このビューに移動してプロジェクト内の変更を追跡する方法を詳しく説明します。

> **Note**
>
> CodeCatalyst コミットによるデプロイステータスの追跡はリポジトリでのみサポートされています。GitHub この機能はリポジトリでは使用できません。

デプロイ状況をコミットごとに追跡するには

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、[トラッキングを変更] 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から、リリースステータスを表示したい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. [変更を表示] を選択します。

   コミットのリストが表示されます。

   各コミットについて、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コミット情報。詳細については、「のソースリポジトリ 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、「環境を使用する」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、「のワークフローを使用したテスト CodeCatalyst」を参照してください。

   > **Note**
   >
   > ソフトウェアコンポジション分析 (SCA) の結果は表示されません。

6. (オプション) 最新のデプロイメント、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情報を表示するには、そのコミットの [詳細を表示] を選択します。

でのワークフローを使用した構築 CodeCatalyst

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いいですか?
- 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を表示する
- チュートリアル:Amazon S3 へのアーティファクトのアップロード

アプリケーションを構築するにはどうすればいいですか?

でアプリケーションまたはリソースをビルドするには CodeCatalyst、まずワークフローを作成し、次に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、単体テストを実行し、すぐにデプロイできるアーティファクトを生成するワークフローの構成要素です。

CodeCatalyst コンソールのビジュアルエディターまたは YAML エディターを使用して、ビルドアクションをワークフローに追加します。

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 (高レベルのタスク)

1. では CodeCatalyst、ビルド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、「でのソースリポジトリの操作 CodeCatalyst」を参照してください。
2. では CodeCatalyst、ワークフローを作成します。ワークフローでは、アプリケーションのビルド、テスト、デプロイの方法を定義します。詳細については、「でのワークフロー入門 CodeCatalyst」を参照してください。
3. (オプション) ワークフローには、ワークフローが自動的に開始される原因とな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、「トリガーの使用」を参照してください。
4. ワークフローには、アプリケーションまたはリソースの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、ビルドアクションにユニットテストを実行させたり、レポートを生成したり、アプリケーションをデプロイしたりすることもできます。これらの目的でテストアクションやデプロイアクションを使用したくない場合は、テストアクションと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) ワークフローに、アプリケーションまたはリソースをテストしてデプロイするためのテストアクションとデプロイアクションを追加します。あらかじめ設定された複数のアクションから選択して、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できます。詳細については、「のワークフローを使用したテスト CodeCatalyst」および「でのワークフローを使用したデプロイ CodeCatalyst」を参照してください。
6. ワークフローは手動で、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、デプロイの各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ことには以下の利点があります。

- 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、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、「コンピュートでの作業」を参照してください。
- すぐに使える — CodeCatalyst 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、あらかじめパッケージされたランタイム環境の Docker イメージが含まれています。これらのイメージには、や Node.js などのアプリケーションの構築に役立つツールがあらかじめ設定されています。 AWS CLI パブリックレジストリーまたはプライベートレジストリーから提供したビルドイメージを使用するように設定できます CodeCatalyst 。詳細については、「ランタイム環境の Docker イメージの操作」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、 CodeCatalyst代わりにデプロイアクションの使用を検討してください。デプロイアクションは、 behind-the-scenesビルドアクションを使用する場合は手動で記述しなければならない構成を実行します。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、「What is CodeBuild?」を参照してください。 。

トピック

- ビルドアクションの追加
- ビルドアクションの結果を表示する
- チュートリアル:Amazon S3 へのアーティファクトのアップロード

ビルドアクションの追加

以下の手順を使用して、 CodeCatalyst ワークフローにビルドアクションを追加します。

Visual

ビジュアルエディターを使用してビルド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. 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ビルドとテストアクションのリファレンス。このリファレンスには、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) の詳細情報が記載されています。
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. [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ビルド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. 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. 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、ビルドとテストアクションのリファレンスに記載されています。
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. [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。

アクション定義を構築します。

ビルドアクションは、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、ビルドとテストアクションのリファレンスワークフロー定義リファレンスのを参照してください。

ビルドアクションの結果を表示する

以下の手順に従って、生成されたログ、レポート、変数を含むビルドアクションの結果を表示します。

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 (Build など) を選択します。
4. ビルド実行のログを表示するには、「Logs」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拡張できます。
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、「テストレポートタイプ」を参照してください。
6. ビルドアクションに使用された構成を表示するには、「構成」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、「変数の操作」を参照してください。

チュートリアル:Amazon S3 へのアーティファクトのアップロード

このチュートリアルでは、 CodeCatalyst いくつかのビルドアクションを含む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を学習します。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では、`Hello.txt`と 2 つのファイルが生成され`Goodbye.txt`、ビルドアーティファクトにバンドルされます。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。ソースリポジトリにコミットをプッシュするたびに実行されるようにワークフローを設定します。

トピック

前提条件

開始するには、以下が必要です。

- CodeCatalyst AWS アカウントが接続されたスペースが必要です。詳細については、「AWS Builder ID ユーザーをサポートするスペースの作成」を参照してください。
- スペースには、以下の名前の空の「ゼロから開始」 CodeCatalyst プロジェクトが必要です。

  ```
  codecatalyst-artifact-project
  ```

  詳細については、「Amazon で空のプロジェクトを作成する CodeCatalyst」を参照してください。
- プロジェクトには、 CodeCatalyst 次のような環境が必要です。

  ```
  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 アカウントをそれConnect。

詳細については、「環境を使用する」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、 AWS IAM ロールを作成します。このロールは後でワークフロー内のビルドアクションに割り当てます。このロールは、 CodeCatalyst AWS ビルドアクションに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 に書き込む権限を付与します。この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。次の手順に示す権限と信頼ポリシーが適用されていることを確認してください。

IAM ロールの詳細については、『AWS AWS Identity and Access Management ユーザーガイド』の「IAM ロール」を参照してください。

ビルドロールを作成するには

1. ロールのポリシーを次のように作成します。
   a. AWSにサインインします。
   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**
>
> 初めてロールを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、リソース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] に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、アクセス権限ポリシーが作成されました。

2. ビルド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. 「アクセス権限ポリシー」で、codecatalyst-s3-build-policy該当するチェックボックスを検索して選択します。

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権限ポリシーを含む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.txtGoodbye.txtと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 [バケットを作成] 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 がサポートす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. ページの下部にある [バケットを作成] を選択します。
6. 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 **codecatalyst-artifact-bucket** リージョンという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/ 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] に、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、順次実行される次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、ソースリポジトリに変更を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の使用。
- GenerateFiles— On trigger GenerateFiles というビルドアクションは、と 2 Hello.txt つのファイルを作成しGoodbye.txt、codecatalystArtifactそれらをという出力アーティファクトにパッケージ化します。
- 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、aws s3 synccodecatalystArtifactソースリポジトリとにあるファイルを Amazon S3 AWS CLI バケットにアップロードするコマンドが実行されます。Upload AWS CLI CodeCatalystはコンピュートプラットフォームにあらかじめインストールされ、設定されているので、インストールや設定は不要です。

  CodeCatalyst コンピュートプラットフォームにあらかじめパッケージされている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 イメージの操作のコマンドについて詳しくは、『*aws s3 sync*コマンドリファレンス』 AWS CLIの「sync」を参照してください。AWS CLI

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でのワークフローを使用した構築 CodeCatalyst。

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
```

上のコードでは、以下を置き換えてください。

- *codecatalyst-artifact-environment*で作成した環境の名前を前提条件。
- *codecatalyst-account-connection*で作成したアカウント接続の名前で前提条件。
- *codecatalyst-s3-build-role* に、で作成したビルドロールの名前を付けてください。ステップ 1: AWS ロールを作成する
- *codecatalyst-artifact-bucket*で作成した Amazon S3 の名前を使用してくださいステップ 2: Amazon S3 バケットを作成する。

このファイル内のプロパティについては、を参照してくださいビルドとテストアクションのリファレンス。

5. (オプション) コミットする前に [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。
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] は、デフォルトのままにします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] には、を選択しますcodecatalyst-artifact-source-repository。

   d. [ブランチ名] には [main] 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。ワークフローの上部に定義されているトリガーにより、ワークフローの実行が自動的に開始されます。具体的には、codecatalyst-artifact-workflow.yamlソースリポジトリにファイルをコミット (およびプッシュ) すると、トリガーによってワークフローの実行が開始されます。

実行中のワークフローを表示するには

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. を選択するとGenerateFiles、最初のビルドアクションの進行状況が表示されます。
4. [Upload] を選択すると、2 番目のビルドアクションの進行状況が表示されます。
5. Upload アクションが終了したら、次の操作を行います。

   - ワークフローが正常に実行されたら、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、[Logs] を選択して問題のトラブルシューティングを行います。

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、*codecatalyst-artifact-bucket*バケットを調べます。これで、以下のファイルとフォルダが含まれているはずです。

```
.
|– .aws/
|– .git/
|Goodbye.txt
|Hello.txt
|REAME.md
```

Goodbye.txtHello.txtcodecatalystArtifactおよびファイルはアー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/、README.mdおよびファイルは、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

CodeCatalyst AWS クリーンアップして、これらのサービスに課金されないようにしてください。

でクリーンアップするには CodeCatalyst

1. https://codecatalyst.aws/ CodeCatalyst で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。

を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。

のワークフローを使用したテスト CodeCatalyst

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、ソフトウェアコンポジション分析、およびスタティック分析レポートを作成するワークフローアクションです。 CodeCatalyst これらのレポートはコンソールに表示されます。

トピック

テストレポートタイプ

Amazon CodeCatalyst テストアクションは、次の種類の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例:レポートの設定。

トピック

テストレポート

では CodeCatalyst、ビルド中に実行される単体テスト、統合テスト、システムテストを設定できます。その後 CodeCatalyst 、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して、テストに関する問題のトラブルシューティングに役立てることができます。複数のビルド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失敗率を表示し、ビルドの最適化に役立てることができます。

以下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、テスト用のコードカバレッジレポートを生成できます。 CodeCatalyst 以下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

テストの対象となるステートメントの数を測定します。ステートメントは 1 つの命令で、コメントは含まれません。

```
line coverage = (total lines covered)/(total number of lines)
```

ブランチカバレッジ

`if`OR `case` 文のような制御構造で考えられるすべての分岐のうち、テストの対象となる分岐の数を測定します。

```
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、.json ではなく simplecov によって生成されます)
- クローバー XML (バージョン 3、.xml)
- XML (.xml) カバレッジ
- CLOV (.info)

ソフトウェア・コンポジション分析レポート

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脆弱性をチェックできます。さまざまな重大度の脆弱性とその修正方法を詳述した SARIF レポートを検出して解析できます。重大度が最も高いものから最も低いものまで、有効な重大度の値は、、、、です。`CRITICAL` `HIGH` `MEDIUM` `LOW` `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)

スタティック分析レポート

静的分析 (SA) レポートを使用して、ソースレベルのコード欠陥を特定できます。では CodeCatalyst、コードをデプロイする前にコード内の問題を解決するのに役立つ SA レポートを生成できます。これらの問題には、バグ、セキュリティ上の脆弱性、品質上の問題、その他の脆弱性が含まれます。重大度が最も高いものから最も低いものまで、有効な重要度の値は`CRITICAL`、、、`HIGH`、`MEDIUM`、`LOW`、および`INFORMATIONAL`です。

CodeCatalyst 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る可能性のあるバグをいくつか特定します。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。バグの例を以下に示します。

```
// 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;
}
```

セキュリティ上の脆弱性

ソースコードに見つかった可能性のあるセキュリティ上の脆弱性をいくつか特定します。これらのセキュリティ脆弱性には、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。

品質上の問題

ソースコードに見つかる可能性のある品質上の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。品質問題の例を以下に示します。

```
// 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

ソースコードに見つかった可能性のあるその他の脆弱性をいくつか特定します。

CodeCatalyst 次の SA レポートファイル形式をサポートします。

- PyLint (.py)
- ESLint (.js、.jsx、.ts、.tsx)
- SARIF (.sarif、.json)

テストアクションの追加

以下の手順に従って、テストアクションをワークフローに追加します。

テストアクションを追加するには

- 「ビルドアクションの追加」の手順に従います ビルドアクションとテストアクションを追加する手順はまったく同じです。

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 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、ビルドとテストアクションのリファレンスワークフロー定義リファレンスのを参照してください。

レポートの設定

このセクションでは、品質レポートの設定方法について説明します。

トピック

自動検出レポートと手動レポートの設定

自動検出を有効にすると、アクションに渡されたすべての入力と、 CodeCatalyst アクション自体によって生成されたすべてのファイルを検索して、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）、および静的分析（SA）レポートを探します。これらのレポートはそれぞれで表示および操作できます。 CodeCatalyst

どのレポートを生成するかを手動で設定することもできます。生成するレポートの種類とファイル形式を指定できます。詳細については、「テストレポートタイプ」を参照してください。

レポートの達成基準の設定

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。

達成基準は、レポートが合格か不合格かを決定する閾値です。 CodeCatalyst 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その成功基準を適用します。次に、達成基準が満たされているかどうか、またどの程度満たされているかがわかります。指定した達成基準を満たさないレポートがある場合、 CodeCatalyst その達成基準を指定したアクションは失敗します。

たとえば、SCA レポートの達成基準を設定した場合、重大度が最も高いものから最も低いものまでの有効な脆弱性の値は`CRITICAL`、、、`HIGH``MEDIUM``LOW`、`INFORMATIONAL`です。重大度レベルで 1 つの脆弱性をスキャンするように条件を設定した場合、`HIGH`重大度の脆弱性が少なくとも 1 つ存在するか、またはまったくないが、`HIGH`重大度レベルの脆弱性が少なくとも 1 つ (たとえば、`HIGH`深刻度が 1 つの脆弱性など) 場合、レポートは失敗します。`CRITICAL`

達成基準を指定しない場合、

- CodeCatalyst 未加工のレポートに基づいて生成されたレポートには、達成基準は表示されません。
- 達成基準は、関連するワークフローアクションが合格か失敗かを判断するのには使用されません。

Visual

達成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。これは達成基準を適用したいレポートです。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 [レポートの手動設定] で、[達成基準] を選択します。

   達成基準が表示されます。以前の選択内容に応じて、次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。

   合格率

   関連するレポートに合格とマークされるためには、 CodeCatalystテストレポート内のテストの合格率を指定します。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に含まれる行のうち、対象レポートが合格とマークされるためにカバーする必要がある行の割合を指定します。 CodeCatalyst 有効な値には 10 進数が含まれます。例: `50`、`60.5`。ライン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に含まれるブランチのうち、対象レポートが合格とマークされ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てください。 CodeCatalyst 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大度を指定して、 CodeCatalyst 関連するレポートに合格とマークしてください。脆弱性を指定するには、以下を指定する必要があります。

- カウントに含めたい脆弱性の最小重要度。有効な値は、重大度が最も高いものから最も低いものまでCRITICAL、、HIGH、MEDIUMLOW、INFORMATIONALです。

  たとえば、選択した場合はHIGH、HIGHCRITICAL脆弱性が集計されます。
- 指定した重要度で許可したい脆弱性の最大数。この数を超えると、 CodeCatalyst レポートは失敗とマークされます。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・コンポジション分析レポート

バグ

CodeCatalyst 関連するレポートに合格とマークされる SA レポートで許可されるバグの最大数と重大度を指定してください。バグを指定するには、以下を指定する必要があります。

- カウントに含めたいバグの最低重要度。重大度が最も高いものから最も低いものまで、有効な値はCRITICAL、HIGH、MEDIUM、LOW、INFORMATIONALです。

  たとえば、選択した場合HIGH、HIGHCRITICALおよびのバグが集計されます。
- 指定した重大度で許可したいバグの最大数。この数を超えると、 CodeCatalyst レポートは失敗とマークされます。有効な値は整数です。

バグ基準は ESLint SA レポートにのみ適用されます。 PyLint SA レポートの詳細については、を参照してください。スタティック分析レポート

セキュリティ上の脆弱性

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大度を指定して、 CodeCatalyst 関連するレポートに合格とマークしてください。セキュリティ上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。

- カウントに含めたいセキュリティ脆弱性の最小重要度。重大度が最も高いものから最も低いものまで、有効な値はCRITICAL、、HIGH、MEDIUMLOW、INFORMATIONALです。

  たとえば、選択した場合HIGHHIGH、CRITICALセキュリティの脆弱性が集計されます。
- 指定した重大度のセキュリティ脆弱性の許容数の上限。この数を超えると、 CodeCatalyst レポートは失敗とマークされます。有効な値は整数です。

セキュリティ脆弱性基準は ESLint SA PyLint 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。スタティック分析レポート

品質に関する問題

SA レポートで許容される品質問題の最大数と重大度を指定して、 CodeCatalyst 関連するレポートに合格とマークしてください。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。

- カウントに含めたい品質問題の最小重要度。重大度が最も高いものから最も低いものまで、有効な値はCRITICAL、HIGH、MEDIUM、LOW、INFORMATIONALです。

  たとえば、選択した場合はHIGHHIGH、CRITICAL品質に関する問題が集計されます。
- 指定した重要度で許可したい品質問題の最大数。この数を超えると、 CodeCatalyst レポートは失敗とマークされます。有効な値は整数です。

品質問題基準は ESLint SA レポートにのみ適用されます。 PyLint SA レポートの詳細については、を参照してください。スタティック分析レポート

8. [Commit] (コミット) を選択します。
9. CodeCatalyst ワークフローを実行して未加工のレポートに達成基準を適用させ、CodeCatalyst 達成基準情報を含む関連レポートを再生成します。詳細については、「ワークフロー実行の開始」を参照してください。

YAML

達成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。これは達成基準を適用したいレポートです。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、Reportsまたはセクションに、、、、PassRateLineCoverageBranchCoverageVulnerabilitiesStaticAnalysisBugよびプロパティとともにプロパティを追加します。SuccessCriteria

   これらの各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とテストアクションのリファレンス。
8. [Commit] (コミット) を選択します。
9. CodeCatalyst ワークフローを実行して未加工のレポートに達成基準を適用させ、CodeCatalyst 達成基準情報を含む関連レポートを再生成します。ワークフローの開始に関する詳細については、を参照してくださいワークフロー実行の開始。

例:レポートの設定

次の例は、テストレポート、コードカバレッジレポート、ソフトウェアコンポジション分析レポート、およびスタティック分析レポートの 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
```

ソフトウェアコンポジション分析レポートとスタティック分析レポートの SARIF サポート

SARIF (スタティック分析結果交換フォーマット) は、のソフトウェアコンポジション分析レポートとスタティック解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、スタティック分析レポートで SARIF を手動で構成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 次の SARIF プロパティをサポートしており、これらを使用して分析結果をレポートに表示する方法を最適化できます。

トピック

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 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 回以上の実行の配列が含まれ、各実行は分析ツールの 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	toolComponent 分析ツールを説明するオブジェクト。
tool.name	[いいえ]	分析を行うために使用す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]	ツールが分析したアーティファクトの総数。
rules[]	はい	reportingDescriptor ルールを表すオブジェクトの配列。分析ツールはこれらのルールに基づいて、分析対象のコード内の問題を検出します。

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固有識別子。

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]	規則の簡潔な説明。 最大長:3,000 文字
fullDescription.text	[いいえ]	ルールの詳細な説明。 最大長:3,000 文字
helpUri	[いいえ]	ルールの主要文書の絶対 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]	スキャン結果にスコアが付けられたかどうかを示すフラグ。
properties.score.severity	[いいえ]	結果の重大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]	共通脆弱性評価システム v3.1 の質的重要度評価
properties.cvssv3_baseScore	[いいえ]	CVSS v3 の基本スコアは 0.0 ~ 10.0 の範囲です。
properties.cvssv2_severity	[いいえ]	CVSS v3 の値が使用できない場合は、CVSS v2 の値を検索します。 CodeCatalyst

名前	必要	説明
properties.cvssv2_score	[いいえ]	CVSS v2 の基本スコアは 0.0 から 10.0 の範囲です。
properties.severity	[いいえ]	結果の重大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	結果を参照するため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	rules[]ツールコンポーネント内の関連ルールのインデックス。
message.text	はい	結果を説明し、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 から 100.0 までの値。このスケールの値は 0.0 が最低優先度で、100.0 が最高優先度です。
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]	スキャン結果にスコアが付けられたかどうかを示すフラグ。
properties.score.severity	[いいえ]	結果の重大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]	共通脆弱性評価システム v3.1 の質的重要度評価
properties.cvssv3_baseScore	[いいえ]	CVSS v3 の基本スコアは 0.0 ~ 10.0 の範囲です。
properties.cvssv2_severity	[いいえ]	CVSS v3 の値が使用できない場合は、CVSS v2 の値を検索します。 CodeCatalyst
properties.cvssv2_score	[いいえ]	CVSS v2 の基本スコアは 0.0 から 10.0 の範囲です。
properties.severity	[いいえ]	結果の重大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。指定した場所ごとに変更を加えるだけで問題を解決できない場合を除いて、1 つの場所だけを含めてください。CodeCatalyst 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]	調査結果で参照されているその他の場所のリスト。 locationオブジェクトの最大数:50
fixes[]	[いいえ]	fixスキャンツールによって提供される推奨事項を表すオブジェクトの配列。CodeCatalyst fixes配列の最初のレコメンデーションを使用します。

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。
logicalLocations[]	[いいえ]	アーティファクトに関係なく名前で記述された口ケーションのセット。

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 (通常はリポジトリ内にあるか、ビルド中に生成されるファイル) の場所を示す URI。
fileLocation.uri	[いいえ]	ファイルの場所を示すフォールバック URI。artifactLocation.uri　空が返された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]	各結果に対する推奨事項を表示するメッセージ。 最大長:3,000 文字
artifactChanges.[0].artifactLocation.uri	[いいえ]	更新が必要なアーティファクトの場所を示す URI。

を使用する universal-test-runner

universal-test-runner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。このツールには、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が備わっています。 universal-test-runner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でテストを実行します。 universal-test-runner以下のフレームワークをサポートします。

- Gradle
- Jest
- Maven
- パイテスト
- .NET

universal-test-runner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、universal-test-runner高度なテスト機能を有効にするために手動でインストールする必要があります。そのためには、Node.js (14 以上) をイメージにインストールし、universal-test-runnernpmシェルコマンドを使用してインストールします。- Run: npm install -g @aws/universal-test-runnerシェルコマンドを使用して Node.js をコンテナーにインストールする方法の詳細については、「Node Version Manager のインストールと更新」を参照してください。

詳細についてはuniversal-test-runner、「とは universal-test-runner?」を参照してください。

Visual

universal-test-runner ビジュアルエディターで使うには

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 [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、選択したサポートされているフレームワークで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。たとえば、サポートされているフレームワークを使用するには、Run次のような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の提供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. [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 universal-test-runner エディターで使用するには

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 [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。たとえば、サポートされているフレームワークを使用するには、Run次のような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の提供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. [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。

ベストプラクティス

が提供するテスト機能を使用するときは 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

- オートディスカバリー
- 成功基準
- パスを含める/除外する

オートディスカバリー

でアクションを設定すると CodeCatalyst、オートディスカバリーにより、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、 CodeCatalystそれらから関連するレポートを生成できます。自動検出により、検出された出力の名前やパスが変更された場合でも、レポートは引き続き生成されます。新しいファイルが追加されると、 CodeCatalyst自動的に検出され、関連するレポートが作成されます。ただし、自動検出を使用する場合は、この機能の以下の点を考慮に入れることが重要です。

- アクションで自動検出を有効にすると、自動的に検出された同じタイプのレポートはすべて同じ達成基準を共有します。たとえば、最低合格率などの共通の基準は、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。同じ種類のレポートに異なる基準が必要な場合は、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を検索することもでき、達成基準が設定されていると、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、除外パスの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、毎回同じレポートリストが生成されるとは限りません。特定のレポートを常に作成したい場合は、レポートを明示的に設定する必要があります。たとえば、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しないため、テストレポートは生成されず、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させたい場合は、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 (インクルード**/*) 全体で自動検出を使用してください。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に対してレポートが生成されます。

詳細については、「レポートの設定」を参照してください。

成功基準

達成基準を設定することで、レポートに品質閾値を適用できます。たとえば、2 つのコードカバレッジレポートが自動検出され、1 つはラインカバレッジ 80%、もう 1 つはラインカバレッジが 60% の場合、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、最初のレポートは合格し、2 番目のレポートは失敗し、アクション全体が失敗することになります。ワークフローの妨げにならないようにするには、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てください。
- 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、両方のレポートが合格し、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% を下回ら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ーを使用するか、各レポートに明示的な YAML セクションとパスを追加して、一方または両方のレポート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達成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、「レポートの達成基準の設定」を参照してください。

パスを含める/除外する

アクション結果を確認する場合、 CodeCatalyst とを設定することで生成されるレポートのリストを調整できます。`IncludePaths` `ExcludePaths`

- `IncludePaths` CodeCatalyst レポートを検索するときに含めるファイルとファイルパスを指定するのに使用します。たとえば、を指定すると`"/test/report/*"`、 CodeCatalyst `/test/report/`アクションが使用するビルドイメージ全体を検索してディレクトリを探します。そのディレクトリが見つかると、 CodeCatalyst そのディレクトリでレポートを検索します。

  > **Note**
  >
  > 手動で設定したレポートでは、1 `IncludePaths` つのファイルと一致するグロブパターンである必要があります。

- CodeCatalyst レポートを検索する際に除外するファイルとファイルパスを指定するのに使用しますExcludePaths。たとえば、を指定した場合`"/test/reports/**/*"`、 CodeCatalyst/`test/reports/`ディレクトリ内のファイルは検索されません。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。

考えられるグロブパターンの例を以下に示します。

パターン	説明
`*.*`	現在のディレクトリ内のドットを含むすべてのオブジェクト名にマッチします。
`*.xml`	現在のディレクトリの末尾にあるすべてのオブジェクト名と一致します。`.xml`

パターン	説明
*.{xml,txt}	現在のディレクトリ内の、.xmlまたはで終わるすべてのオブジェクト名にマッチします。.txt
**/*.xml	で終わるすべてのディレクトリのオブジェクト名と一致します。.xml
testFolder	testFolder という名前のオブジェクトにマッチし、ファイルとして扱います。
testFolder/*	サブフォルダの 1 つのレベルのオブジェクト (例:) と照合します。testFolder testFolder/file.xml
testFolder/*/*	サブフォルダの 2 つのレベルにあるオブジェクト (例:) を照合します。testFolder 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 グロブ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より優先されます。IncludePathsIncludePathsExcludePaths両方に同じフォルダーが含まれている場合、そのフォルダーのレポートはスキャンされません。

テストでの作業

このセクションでは、テストアクションを表示して設定する方法について説明します。

トピック

- テストアクションの結果を表示する
- 失敗したテストをスキップする
- テストケースの再試行

テストアクションの結果を表示する

以下の手順に従って、生成されたログ、レポート、変数を含むテストアクションの結果を表示します。

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 (Test など) を選択します。
4. アクションによって生成されたログを表示するには、[Logs] 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。ログは必要に応じて展開したり折りたたんだりできます。
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」を選択します。詳細については、「テストレポートタイプ」を参照してください。
6. テストアクションに使用した構成を表示するには、「構成」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、「変数の操作」を参照してください。

失敗したテストをスキップする

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ても、アクション内の後続のテストコマンドを実行できるようにしたい場合があります。たとえば、次のコマンドでは、test2test1失敗しても常に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 CodeCatalyst はワークフローアクションを停止し、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そのためには、2>/dev/nullコマンドに追加してください。この変更を加えた場合、前の例は以下のようになります。

```
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考慮されますが、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は、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確認し、壊れた依存関係を処理したり、ワークフローを再実行したりするなど、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、universal-test-runner選択したテストケースのみを再試行する機能が組み込まれています。1 回の操作で選択したテストケースを 1 セットだけ再試行でき、テストレポート 1 つにつき 5 回しか再試行できません。詳細については、「を使用する universal-test-runner」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てください。

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。名前、ステータス、リポジトリ、ブランチ、またはレポートの種類でフィルタリングできます。
3. レポート名の下にある [Results] 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] を選択してから [選択したテストケース] を選択します。
5. 再試行が終了したら、バナーの [Refresh] を選択し、更新された結果を確認します。

でのワークフローを使用したデプロイ CodeCatalyst

CodeCatalyst ワークフローを使用すると、Amazon ECS などのさまざまなターゲットにアプリケーションやその他のリソースをデプロイできます。 AWS Lambda

トピック

- アプリケーションをデプロイする方法を教えてください。
- デプロイアクションのリスト。
- デプロイアクションの利点
- デプロイアクションの代替手段
- 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 へのアプリケーションのデプロイ
- チュートリアル:Amazon EKS へのアプリケーションのデプロイ
- 「 AWS CloudFormation デプロイスタック」アクションの追加
- 「Amazon ECS にデプロイ」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 CDK デプロイ」アクションの追加
- デプロイでの作業

アプリケーションをデプロイする方法を教えてください。

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、デプロイする内容、デプロイする場所、デプロイ方法 (ブルー/グリーンスキームを使用するなど) を定義するワークフローの構成要素です。デプロイアクションは、 CodeCatalyst コンソールのビジュアルエディターまたは YAML エディターを使用してワークフローに追加します。

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 (高レベルのタスク)

1. CodeCatalyst プロジェクトでは、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、「でのソースリポジトリの操作 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。ワークフローは、アプリケーションのビルド、テスト、デプロイの方法を定義する場所です。詳細については、「でのワークフロー入門 CodeCatalyst」を参照してください。
3. ワークフローには、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、トリガーの使用、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. ワークフローでは、デプロイアクションを追加します。 CodeCatalyst提供されている複数のデプロイアクションの中から、Amazon ECS などのさまざまなターゲットへのアプリケーションへのデプロイアクションを選択できます。(GitHub ビルドアクションまたは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。 GitHub ビルドアクションと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代替手段。)
5. ワークフローは手動で、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、デプロイの各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。

以下のデプロイアクションを使用できます。

- AWS CloudFormation デプロイスタック — このアクションは、 CloudFormation AWS AWS CloudFormationAWS Serverless Application Model 指定したテンプレートまたはテンプレートに基づいてスタックを作成します。詳細については、「「 AWS CloudFormation デプロイスタック」アクションの追加」を参照してください。
- Amazon ECS にデプロイ — 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、「「Amazon ECS にデプロイ」アクションの追加」を参照してください。
- Kubernetes クラスターへのデプロイ — このアクションは、Amazon Elastic Kubernetes Service スクラスターにアプリケーションをデプロイします。詳細については、「「Kubernetes クラスターへのデプロイ」アクションの追加」を参照してください。
- AWS CDK デプロイ — このアクションはアプリケーションをにデプロイします。AWS CDK AWS 詳細については、「「AWS CDK デプロイ」アクションの追加」を参照してください。

> **Note**
>
> CodeCatalyst リソースをデプロイできるアクションは他にもありますが、デプロイ情報が Environments 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」ページとデプロイメントの表示について詳しくは、「」環境を使用する と デプロイでの作業「」を参照してください。

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ことには以下の利点があります。

- デプロイ履歴 — デプロイの履歴を表示して、デプロイしたソフトウェアの変更を管理したり伝達したりするのに役立ちます。
- 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できます。
- ロールバック — エラーが発生した場合、デプロイメント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有効にするアラームを設定することもできます。
- 監視 — ワークフローのさまざまな段階でデプロイが進むのを監視します。
- CodeCatalyst 他の機能との統合 — ソースコードを保存し、ビルド、テスト、デプロイをすべて1つのアプリケーションから行えます。

デプロイアクションの代替手段

デプロイアクションは必ずしも使用しなくてもかまいませんが、前のセクションで説明した利点があるため推奨されます。代わりに、CodeCatalyst 以下のアクションを使用できます。

- ビルドアクション。

  通常、ビルドアクションは、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や、デプロイメント手順をより細かく制御したい場合に使用します。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について詳しくは、を参照してくださいでのワークフローを使用した構築 CodeCatalyst。

- GitHub アクション。

  GitHub CodeCatalyst ワークフロー内でアクションを使用して、アプリケーションとリソースを (CodeCatalystアクションの代わりに) デプロイできます。 GitHub CodeCatalyst ワークフロー内で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。 GitHub アクションの追加

CodeCatalyst ワークフローを使用したくない場合は、 AWS 以下の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 CodeDeploy?」を参照してください。
- AWS CodeBuild そして AWS CodePipeline —「とは AWS CodeBuild?」を参照してください。 そして、何ですか AWS CodePipeline？
- AWS CloudFormation —「AWS CloudFormationとは何か」を参照してください。

CodeDeploy、、 CodeBuild CodePipeline、 CloudFormation のサービスを使用して、複雑な企業への導入を行います。

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、ワークフローとを使用してサーバーレスアプリケーションをビルド、テスト、デプロイする方法を学習します。 AWS CloudFormation

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 Web アプリケーションです。 AWS Lambda これは関数と Amazon API Gateway で構成されており、の

拡張である AWS Serverless Application Model (AWS SAM) AWS CloudFormationを使用して構築します。

トピック

前提条件

開始する前に:

- CodeCatalyst AWS アカウントが接続されたスペースが必要です。詳細については、「AWS Builder ID ユーザーをサポートするスペースの作成」を参照してください。
- スペースには、以下の名前の空の「ゼロから開始」 CodeCatalyst プロジェクトが必要です。

  ```
  codecatalyst-cfn-project
  ```

  詳細については、「Amazon で空のプロジェクトを作成する CodeCatalyst」を参照してください。
- プロジェクトには、 CodeCatalyst 次のような環境が必要です。

  ```
  codecatalyst-cfn-environment
  ```

  この環境を以下のように設定します。
  - 「非実稼働」など、どのタイプでも選択できます。
  - AWS アカウントをそれConnect。

  詳細については、「環境を使用する」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. 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] を選択します。 CodeCatalyst
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] に、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。

- Deploy role — CodeCatalyst AWS CloudFormation サーバーレスアプリケーションをデプロイするアカウントとサービスにアクセスするための Deploy AWS CloudFormation stack アクション権限を付与します。Deploy AWS CloudFormation stack 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 CodeCatalyst に書き込むためのビルドアクション権限を付与します。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 — AWS SAM 後で指定するテンプレートで指定されているリソースを読み取り、CloudFormation 変更する権限を付与します。また、 CloudWatchに権限を付与します。

IAM ロールの詳細については、『AWS Identity and Access Management ユーザーガイド』の「IAM ロール」を参照してください。

> **Note**
>
> 時間を節約するために、前述の 3 つのロールの代わりに、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用の CodeCatalystWorkflowDevelopmentRole-***spaceName*** ロールの作成」を参照してください。CodeCatalystWorkflowDevelopmentRole-*spaceName*このロールには非常に幅広い権限があり、セキュリティ上のリスクが生じる可能性があることを理解してください。このロールは、セキュリティがそれほど問題になら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 sam-template.yml のワークフローを実行するとファイルが自動的に作成するので、ここで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. ロールのポリシーを次のように作成します。
   a. AWSにサインインします。
   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**
>
> 初めてロールを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、リソース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] に、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、アクセス権限ポリシーが作成されました。

2. 以下のようにデプロイ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. 「アクセス権限ポリシー」で、codecatalyst-deploy-policy該当するチェックボックスを検索して選択します。

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権限ポリシーを含むデプロイロールが作成されました。

3. デプロイロール ARN を次のように取得します。

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 (codecatalyst-deploy-role) を入力します。

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. 上部にある ARN 値をコピーします。

これで、適切な権限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. ロールのポリシーを次のように作成します。

   a. AWSにサインインします。

   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**
>
> 初めてロールを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、リソース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] に、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、アクセス権限ポリシーが作成されました。

2. ビルド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. 「アクセス権限ポリシー」で、codecatalyst-build-policy該当するチェックボックスを検索して選択します。

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権限ポリシーを含むビルドロールが作成されました。

3. ビルドロールの ARN を次のように取得します。

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 (codecatalyst-build-role) を入力します。

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. 上部にある ARN 値をコピーします。

これで、適切な権限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. AWS スタックをデプロイするアカウントを使用してサインインします。
2. IAM コンソール (https://console.aws.amazon.com/iam/) を開きます。
3. スタック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 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」セクションで、CloudFormationドロップダウンリストから選択します。

e. ラジオボタン [CloudFormation] を選択します。

f. 一番下にある [次へ] を選択します。

g. 検索ボックスを使用して以下のアクセス権限ポリシーを探し、それぞれのチェックボックスを選択します。

> **Note**
>
> ポリシーを検索しても表示されない場合は、必ず [フィルターをクリア] を選択してやり直してください。

- CloudWatchFullAccess
- AWS CloudFormationFullAccess
- IAM FullAccess
- AWSラムダ_ FullAccess
- アマゾン API GatewayAdministrator
- アマゾン S3 FullAccess
- アマゾン EC2 ContainerRegistryFullAccess

最初のポリシーでは、 CloudWatch アラーム発生時にスタックロールバックを有効にするためのアクセスを許可します。

残りのポリシーは、 AWS SAM このチュートリアルでデプロイされるスタックのサービスとリソースへのアクセスを許可します。詳細については、『AWS Serverless Application Model 開発者ガイド』の「権限」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. スタックロールの ARN を次のように取得します。

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 (`codecatalyst-stack-role`) を入力します。

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、適切な権限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: AWS ロールをに追加 CodeCatalyst

このステップでは、 CodeCatalyst スペース内のアカウント接続にビルドロール (`codecatalyst-build-role`codecatalyst-deploy-role) とデプロイロール () を追加します。

> **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 デプロイロールとの間ですでに接続が確立された後に、CloudFormation(not CodeCatalyst) AWS がスタックロールを使用するからです。CodeCatalyst スタックロールはへのアクセスには使用されないため AWS、アカウント接続に関連付ける必要はありません。

アカウント接続にビルドロールとデプロイロールを追加するには

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. AWS 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アカウント接続を選択します。
4. 管理コンソールから [AWS ロールを管理] を選択します。

   「Amazon CodeCatalyst スペースに IAM ロールを追加」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。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が設定されたすべての 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`、[Add role] 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] を選択して、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がスペースに追加されました。
8. Amazon CodeCatalyst ディスプレイネーム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の.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 [バケットを作成] 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 がサポートす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. ページの下部にある [バケットを作成] を選択します。

これで、米国西部 (オレゴン) us-west-2 **codecatalyst-cfn-s3-bucket** リージョンという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、 CodeCatalyst 複数のアプリケーションソースファイルをソースリポジトリに追加します。`hello-world`このフォルダーには、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`このフォルダーにはユニットテストが含まれています。フォルダー構造は以下のとおりです。

```
.
|— hello-world
|   |— tests
|      |— unit
|         |— test-handler.js
|   |— app.js
|— .npmignore
|— package.json
|— sam-template.yml
|— setup-sam.sh
```

.np.mignore ファイル

.npmignoreこのファイルには、npm がアプリケーションパッケージから除外すべきファイルとフォルダーが示されています。このチュートリアルでは、npm はフォルダーを除外します。なぜなら、testsこのフォルダーはアプリケーションの一部ではないからです。

.npmignore ファイルを追加するには

1. https://codecatalyst.aws/ でコンソールを開きます。 CodeCatalyst
2. プロジェクトを選択し、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、codecatalyst-cfn-source-repositoryリポジトリを選択します。
5. [ファイル] で [ファイルを作成] を選択します。
6. [ファイル名] に、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、.npmignoreリポジトリのルートにというファイルが作成されました。

パッケージ.json ファイル

package.jsonこのファイルには、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の操作方法や実行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には、test依存関係のリストとスクリプトが含まれています。package.jsonテストスクリプトは次のことを行います。

- テストスクリプトは mocha を使用して、hello-world/tests/unit/で指定されたユニットテストを実行し、その結果を xunit junit.xml レポーターを使用してファイルに書き込みます。

- Istanbul (nyc) を使用して、テストスクリプトは clover 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。詳しくは、Istanbul ドキュメンテーションの「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 [ファイルを作成] 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、package.jsonリポジトリのルート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`このファイルには、Lambda 関数と API Gateway をデプロイし、それらを一緒に設定するための手順が含まれています。テンプレート仕様に準拠しており、AWS Serverless Application Model テンプレート仕様を拡張しています。 AWS CloudFormation

このチュートリアルでは、 AWS SAM 便利な:: Serverless:AWS: Function AWS SAM リソースタイプが提供されているため、 AWS CloudFormation 通常の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、 behind-the-scenes 基本構文を使うために通常書き出さなければならない多くの設定を行います。 CloudFormation たとえば、は Lambda 関数、Lambda 実行ロール、`AWS::Serverless::Function`および関数を開始するイベントソースマッピングを作成します。basic を使用して記述する場合は、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あらかじめ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することもできます。詳細については、「「 AWS CloudFormation デプロイスタック」アクションの追加」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、[ファイルを作成] 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、sam-template.ymlリポジトリのルートフォルダの下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このファイルには、 AWS SAM CLI ユーティリティのダウンロードとインストールの手順が記載されています。ワークフローでは、hello-worldこのユーティリティを使用して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 [ファイルを作成] 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
```

   上記のコードでは、*us-west-2 ####################*。 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、setup-sam.shリポジトリのルート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 app.js 関数コードが含まれています。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 [ファイルを作成] 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、hello-worldという名前のフォルダー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このファイルには、Lambda 関数のユニット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 [ファイルを作成] 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、test-handler.jshello-world/tests/unitフォルダー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。

作業内容を再確認し、すべてのファイルが正しいフォルダに配置されていることを確認してください。フォルダー構造は以下のとおりです。

```
.
|— 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、順番に実行される以下の構成要素で構成されています。

- トリガー — このトリガーは、ソースリポジトリに変更を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、「トリガーの使用」を参照してください。
- テストアクション (Test) — トリガー時に、このアクションはノードパッケージマネージャー (npm) をインストールし、コマンドを実行します。npm run testこのコマンドは、testファイルに定義されているスクリプトを実行するよう npm に指示します。package.json次に、testこのスクリプトはユニットテストを実行し、テストレポート () とコードカバレッジレポート (junit.xml) の 2 つのレポートを生成します。clover.xml詳細については、「パッケージ.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 CodeCatalyst レポートをレポートに変換し、 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のワークフローを使用したテスト CodeCatalyst。

- 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て、Lambda サービスが想定する Amazon S3 バケットにパッケージをコピーします。また、AWS SAM sam-template-packaged.ymlこのアクションはという名前の新しいテンプレートファイルを出力し、という出力アーティファクトに格納します。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でのワークフローを使用した構築 CodeCatalyst。
- 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探し、 AWS SAM その中の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 このテンプレートは、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、次に [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] には [] 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
```

前述のコードでは、以下を置き換えてください。

- *codecatalyst-cfn-environment*のインスタンスは両方とも、ご使用の環境の名前を使用してください。
- のどちらのインスタンスにも、*codecatalyst-account-connection*アカウント接続の表示名が付いています。表示名は数字の場合があります。詳細については、「ステップ 3: AWS ロールをに追加 CodeCatalyst」を参照してください。
- *codecatalyst-build-role*で作成したビルドロールの名前と一緒にステップ 2: AWS ロールを作成する。
- codecatalyst-cfn-s作成した Amazon *S3 ##########* 3 つのバケット。ステップ 4: Amazon S3 バケットを作成する
- *us-west-2* の両方のインスタンスで、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)。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 Region us-west-2 がに設定されていることを前提としています。Amazon S3 でサポートされ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と一緒に。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で前提条件。
- *arn: aws: iam:: 111122223333:role/* に、作成したスタックロールのAmazon リソースネーム (ARN) StackRole を付けてください。ステップ 2: AWS ロールを作成する

> **Note**
>
> ビルド、デプロイ、スタックのロールを作成しないことに決めた場合は、、、、*arn: aws: iam: StackRole 111122223333:role/* をロールの名前または ARN に置き換えてください*codecatalyst-build-role*。*codecatalyst-deploy-role*CodeCatalystWorkflowDevelopmentRole-*spaceName*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。

前に示したコード内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」アクションリファレンス

8. (オプション) コミットする前に [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。
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. [ワークフローのファイル名] は、デフォルトの [] codecatalyst-cfn-workflow のままにします。
    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] には、を選択しますcodecatalyst-cfn-source-repository。
    d. [ブランチ名] には [main] 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。ワークフローの上部に定義されているトリガーにより、ワークフローの実行が自動的に開始されます。具体的には、codecatalyst-cfn-workflow.yamlソースリポジトリにファイルをコミット (およびプッシュ) すると、トリガーによってワークフローの実行が開始されます。

実行中のワークフローを表示するには

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. [Runs] タブを選択します。
4. 「実行 ID」列で、実行 ID を選択します。
5. [Test] を選択してテストの進行状況を確認します。
6. を選択してBuildBackend、ビルドの進行状況を表示します。
7. を選択してDeployCloudFormationStack、デプロイの進行状況を表示します。

   実行詳細の表示について詳しくは、を参照してくださいワークフロー実行の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。

   - ワークフローが正常に実行されたら、次の手順に進みます。
   - BuildBackendテストまたはアクションでワークフローの実行が失敗した場合は、[Logs] を選択して問題のトラブルシューティングを行います。
   - ワークフローの実行がアクションで失敗した場合は、DeployCloudFormationStackデプロイアクションを選択し、次に [サマリー] 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すると、詳細なエラーメッセージが表示されます。ロールバックが発生した場合は、 AWS ワークフローを再実行する前に、`codecatalyst-cfn-stack` AWS CloudFormation コンソールからスタックを削除してください。

デプロイを検証するには

1. デプロイが成功したら、上部近くの水平メニューバーから [変数 (7)] を選択します。(右側のペインで [変数] を選択しないでください)。
2. の横にある `https://` URL をブラウザに貼り付けます。HelloWorldApi

   Lambda 関数からの hello world JSON メッセージが表示されます。これは、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**
>
> いくつか小さな設定を行うだけで、この URL CodeCatalyst をワークフロー図に表示できます。詳細については、「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。

ユニットテストの結果とコードカバレッジを確認するには

1. ワークフロー図で [テスト] を選択し、次に [レポート] を選択します。
2. ユニットテストの結果を表示するか、テスト対象ファイルのコードカバレッジの詳細 (この場合は、app.jsと) を表示するかを選択しますCoverageReporttest-handler.js。TestReport

デプロイされたリソースを確認するには

1. AWS Management Console にサインインし、https://console.aws.amazon.com/apigateway/ にある 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-stackAPI を確認します。API 名はワークフロー定義ファイル (codecatalyst-cfn-workflow.yaml) Configuration/name の値に由来します。
3. https://console.aws.amazon.com/lambda/ AWS Lambda で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 関数を選択してください。codecatalyst-cfn-stack-HelloWorldFunction-*string*
6. API Gateway が関数のトリガーになっていることがわかります。 AWS SAM AWS::Serverless::Functionこのインテグレーションは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: 変更を加える

このステップでは、Lambda ソースコードに変更を加えてコミットします。このコミットにより、新しいワークフローの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ているデフォルトのトラフィックシフト設定を使用する青緑スキームで新しい Lambda 関数がデプロイされます。

Lambda ソースに変更を加えるには

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の変更:

   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 `hello world` **`Tutorial complete!`** 行目をに変更します。

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。名前の変更を反映するように単体テストを更新していないため、この実行は失敗します。

5. ユニットテストを更新してください。

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 `hello world` **`Tutorial complete!`** 行目をに変更します。

d. [コミット] を選択し、もう一度 [コミット] を選択します。

このコミットにより、別のワークフローが実行されます。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`**、次に [実行] を選択します。
8. 最新の実行の実行 ID を選択します。まだ進行中のはずです。
9. [Test] BuildBackend、[] を選択し、ワークフローの実行の進行状況を確認します。DeployCloudFormationStack
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. HelloWorldApiの横にある `https://` URL をブラウザに貼り付けます。

`Tutorial complete!`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メッセージがブラウザーに表示されます。

クリーンアップ

課金されないように、このチュートリアルで使用しているファイルとサービスをクリーンアップしてください。

CodeCatalyst コンソールでクリーンアップするには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. 削除`codecatalyst-cfn-workflow`。

3. 削除codecatalyst-cfn-environment。
4. 削除codecatalyst-cfn-source-repository。
5. 削除codecatalyst-cfn-project。

でクリーンアップするには AWS Management Console

1. 次のようにクリーンアップします。 CloudFormation
   a. https://console.aws.amazon.com/cloudformation AWS CloudFormation で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 と 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 Deploy stack アクションを使用して、CloudFormation AWS CloudFormation サーバーレスアプリケーションをスタックとしてデプロイする方法を学びました。

チュートリアル:Amazon ECS へのアプリケーションのデプロイ

このチュートリアルでは、ワークフロー、Amazon ECS、およびその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を学びます。 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について説明し、次に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、Amazon ECS の完全なセットアップを代行してくれる設計図を使用することもできます。Node.js API をブループリントと一緒に使用するか、Java API AWS Fargate をブループリントと一緒に使用する必要があります。 AWS Fargate詳細については、「ブループリントを使っ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

開始する前に:

- CodeCatalyst AWS アカウントが接続されているスペースが必要です。詳細については、「AWS Builder ID ユーザーをサポートするスペースの作成」を参照してください。
- スペースには、以下の名前の空の「ゼロから開始」 CodeCatalyst プロジェクトが必要です。

```
codecatalyst-ecs-project
```

詳細については、「Amazon で空のプロジェクトを作成する CodeCatalyst」を参照してください。

- プロジェクトには、 CodeCatalyst 次のような環境が必要です。

```
codecatalyst-ecs-environment
```

この環境を以下のように設定します。

- 「非実稼働」など、どのタイプでも選択できます。
- AWS アカウントをそれConnect。

詳細については、「環境を使用する」を参照してください。

ステップ 1: AWS ユーザーを設定し、 AWS CloudShell

このチュートリアルの最初のステップは AWS IAM Identity Center、でユーザーを作成し、 AWS CloudShell そのユーザーとしてインスタンスを起動することです。このチュートリアルの間、CloudShell は開発用コンピューターであり、 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てください。

> **Note**
>
> このチュートリアルでは root 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。そうしないと、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』とAWS CloudShell 『ユーザーガイド』を参照してください。

IAM ID センターのユーザーを作成するには

1. AWS Management Console にサインインし、https://console.aws.amazon.com/singlesignon/ AWS IAM Identity Center のコンソールを開きます。

   > **Note**
   >
   > 必ず、 AWS アカウント CodeCatalystスペースに接続されているを使用してサインインしてください。どのアカウントが接続されているかは、スペースに移動して AWS アカウントタブを選択することで確認できます。詳細については、「AWS Builder ID ユーザーをサポートするスペースの作成」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] に、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を生成する」を選択します。
5. [メールアドレス] と [メールアドレスの確認] に、IAM Identity Center にまだ存在しないメールアドレスを入力します。
6. [名] と [姓] に、次のように入力します。

   ```
   CodeCatalystECSUser
   ```

7. [表示名] には、自動的に生成された名前のままにします。

   ```
   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、「ユーザーを追加」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. [Copy] を選択し、 AWS アクセスポータル 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

この権限セットは後で割り当てます。CodeCatalystECSUser

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. [定義済みの権限セット] を選択し、を選択しますAdministratorAccess。 AWS のサービスこのポリシーはすべてのユーザーに完全な権限を付与します。
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。

権限セットを CodeCatalyst ECSUser 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、 AWS アカウント 現在サインインしているアカウントの横にあるチェックボックスを選択します。
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、CodeCatalystECSUserCodeCatalystECSPermissionSetとをに割り当てて AWS アカウント、まとめることができました。

サインアウトして CodeCatalyst ECSUser 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わかっていることを確認してください。CodeCatalystECSUserこの情報は事前にテキストエディタにコピーしておくべきでした。

> **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、[パスワードをリセット]、[ワンタイムパスワードの生成] を選択します [...] 、パスワードをもう一度リセットすると、情報が画面に表示されます。

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] にパスワードを入力し、[新しいパスワードを設定] 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、 AWS アカウント CodeCatalystECSUserユーザと権限セッ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、CodeCatalystECSUser適切な権限でサインインできました。

AWS CloudShell インスタンスを起動するには

1. CodeCatalystECSUserというわけで、 AWS 上部のナビゲーションバーにあるアイコン
   ( )
   を選択します。

   AWS Management Console のメインページが表示されます。
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お待ちください。

> **Note**
>
> CloudShell アイコンが表示されない場合は、がサポートしているリージョンにいることを確認してください CloudShell。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、 AWS CLI 通常のようにキーや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 サーバーです。

Amazon ECS の詳細については、Amazon エラスティックコンテナサービス開発者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、Amazon ECS AWS Fargate (Fargate) とユーザーに代わって API 呼び出しを行う権限を付与します。

1. 信頼ポリシーの作成:

   a. で AWS CloudShell、以下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 (}) の後ろにカーソルを置きます。

d. **Enter**and キーを押してファイルを保存し**Ctrl+d**、cat を終了します。

2. タスク実行ロールの作成:

```
aws iam create-role \
      --role-name codecatalyst-ecs-task-execution-role \
      --assume-role-policy-document file://codecatalyst-ecs-trust-policy.json
```

3. AWS AmazonECSTaskExecutionRolePolicy管理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. "Arn":ロールの値 (例:) arn:aws:iam::111122223333:role/codecatalyst-ecs-task-execution-role を書き留めます。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が含まれ、後には Hello World アプリケーションが格納されます。

1. ではCodeCatalystECSUser AWS CloudShell、空のクラスターを作成します。

```
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、作成が成功したことが示されます。

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には、取得元の Apache 2.4 Web サーバーの Docker イメージ (httpd:2.4) を実行するように指示されています。 DockerHub

1. のようにCodeCatalystECSUser AWS CloudShell、タスク定義ファイルを作成します。

```
cat > taskdef.json
```

2. 次のコードをプロンプトに貼り付けます。

```
{
    "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"
}
```

上記のコードでは、*arn: aws: iam:: 111122223333:role/*-role を置き換えてください codecatalyst-ecs-task-execution

でメモしたタスク実行ロールの ARN を使用します。タスク実行ロールを作成するには:

3. 最後の中括弧 (}) の後ろにカーソルを置きます。
4. **Enter**and キーを押してファイルを保存し**Ctrl+d**、cat 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. でCodeCatalystECSUser AWS CloudShell、タスク定義を登録します。

```
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るはずです。

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、その後 Hello World アプリケーションのタスク (および関連する Docker コンテナ) を実行します。

1. まだ行っていない場合は`CodeCatalystECSUser`、Amazon Elastic Container Service コンソールに切り替えてください。
2. 先ほど作成したクラスターを選択してください`codecatalyst-ecs-cluster`。
3. [サービス] タブで [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. 次に示す設定以外は、すべてデフォルト設定を維持します。

   b. [Launch type (起動タイプ)] で、[FARGATE] を選択します。

   c. 「タスク定義」の「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、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 1 つの 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 の場合は、任意の VPC を選択します。

   h. [サブネット] では、任意のサブネットを選択します。

      > **Note**
      >
      > サブネットは 1 つだけ指定してください。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、作成してください。Amazon VPC ユーザーガイドの「VPC を作成する」と「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] には [HTTP] を選択し、[ソース] には [任意の場所] を選択します。

j. 一番下にある [作成] を選択します。

k. サービスが作成されるまでお待ちください。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 つのタスクすべての [最終ステータス] 列が [実行中] 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中であることを確認するには

1. 「Tasks」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 イメージを使用して 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ています。

   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。これらすべての項目が揃ったので、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できました。

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に代わるチュートリアルの Docker イメージが格納されます。

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 にイメージリポジトリを作成するには

1. のようにCodeCatalystECSUser、Amazon ECR に空のリポジトリを作成します。 AWS CloudShell

   ```
   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 (例:) を書き留めます。111122223333.dkr.ecr.us-west-2.amazonaws.com/codecatalyst-ecs-image-repo

   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。これらのロールは以下のとおりです。

- ビルドロール — AWS アカウントにアクセスして Amazon ECR と Amazon EC2 に書き込むためのアクセス権限を (ワークフロー内の) CodeCatalyst ビルドアクションに付与します。
- デプロイロール — (ワークフロー内の) CodeCatalyst Deploy to ECS アクションに、 AWS アカウント、Amazon ECS、 AWS およびその他のいくつかのサービスにアクセスする権限を付与します。

IAM ロールの詳細については、ユーザーガイドの「IAM ロール」を参照してください。AWS Identity and Access Management

> **Note**
>
> 時間を節約するために、前述の 2 つのロールの代わりに、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用の CodeCatalystWorkflowDevelopmentRole-***spaceName*** ロールの作成」を参照してください。CodeCatalystWorkflowDevelopmentRole-*spaceName*このロールには非常に幅広い権限があり、セキュリティ上のリスクが生じる可能性があることを理解してください。このロールは、セキュリティがそれほど問題になら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すすめします。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. ロールのポリシーを次のように作成します。
   a. AWSにサインインします。
   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**
>
> 初めてロールを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、リソース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] に、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、アクセス権限ポリシーが作成されました。

2. ビルド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. 「アクセス権限ポリシー」でcodecatalyst-ecs-build-policy、該当するチェックボックスを検索して選択します。

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-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、アクセス権限ポリシーと信頼ポリシーを含むビルドロールが作成されました。

3. ビルドロールの ARN を次のように取得します。

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 (codecatalyst-ecs-build-role) を入力します。

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. 上部にある 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. ロールのポリシーを次のように作成します。

   a. AWSにサインインします。

   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**
>
> 初めてロールを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のワイルドカードを使用してください。リソース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] に、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、アクセス権限ポリシーが作成されました。

2. 以下のようにデプロイ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. 「アクセス権限ポリシー」でcodecatalyst-ecs-deploy-policy、該当するチェックボックスを検索して選択します。

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-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、信頼ポリシーを含むデプロイロールが作成されました。

3. デプロイロール ARN を次のように取得します。

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 (codecatalyst-ecs-deploy-role) を入力します。

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. 上部にある 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

ではCodeCatalystECSUser AWS CloudShell、信頼ポリシーファイルを作成します。

1. ファイルの作成:

```
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 (}) の後ろにカーソルを置きます。
4. **Enter**and キーを押してファイルを保存し**Ctrl+d**、cat 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する:

   a. ではCodeCatalystECSUser AWS CloudShell、ビルドポリシーファイルを作成します。

   ```
   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 (}) の後ろにカーソルを置きます。

   d. **Enter**and キーを押してファイルを保存し**Ctrl+d**、cat を終了します。

2. AWSビルドポリシーを以下に追加します。

```
aws iam create-policy \
    --policy-name codecatalyst-ecs-build-policy \
    --policy-document file://codecatalyst-ecs-build-policy.json
```

3. "arn":コマンド出力の値 (例:) arn:aws:iam::111122223333:policy/codecatalyst-ecs-build-policy を書き留めます。この ARN は後で必要になります。
4. ビルドロールを作成し、それに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、先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. "Arn":ロールの値 (例:) arn:aws:iam::111122223333:role/codecatalyst-ecs-build-role を書き留めます。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する:

   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**
>
> 初めてロールを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、リソースが使用可能になったら、リソース名を使用してポリシーの範囲を絞り込みます。
>
> ```
> "Resource": "*"
> ```

c. 最後の中括弧 (}) の後ろにカーソルを置きます。

d. **Enter**and キーを押してファイルを保存し**Ctrl+d**、cat を終了します。

2. AWSデプロイポリシーを以下に追加します。

```
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、"arn":デプロイポリシーの値 (例:) arn:aws:iam::111122223333:policy/codecatalyst-ecs-deploy-policy を書き留めます。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、それに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。*arn: aws: iam:: 111122223333:policy/ codecatalyst-ecs-deploy-policy* は、先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. "Arn":ロールの値 (例:) arn:aws:iam::111122223333:role/codecatalyst-ecs-deploy-role を書き留めます。この ARN は後で必要になります。

ステップ 5: AWS ロールを追加 CodeCatalyst

このステップでは、 CodeCatalyst スペース内のアカウント接続にビルドロール (codecatalyst-ecs-build-rolecodecatalyst-ecs-deploy-role) とデプロイロール () を追加します。

アカウント接続にビルドロールとデプロイロールを追加するには

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. AWS 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アカウント接続を選択します。
4. 管理コンソールから [AWS ロールを管理] を選択します。

   「Amazon CodeCatalyst スペースに IAM ロールを追加」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。リストには、codecatalyst-runner.amazonaws.comcodecatalyst.amazonaws.comおよびサービスプリンシパルを含む信頼ポリシーが設定されたすべての 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、[Add role] を選択します。

> **Note**
>
> 表示される場合はThe security token included in the request is invalid、適切な権限がないことが原因である可能性があります。この問題を解決するには、サインアウトして、 AWS CodeCatalystスペースを作成したときに使用したアカウントでサインインし直してください。 AWS

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-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がスペースに追加されました。

8. Amazon CodeCatalyst ディスプレイネーム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格納されます。

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/ 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] に、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 CodeCatalyst のソースファイルをリポジトリに追加します。`codecatalyst-ecs-source-repository`これらには以下のものが含まれます。

- `index.html`ファイル — Hello World メッセージをブラウザに表示します。
- Dockerファイル — 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 コマンドを記述します。
- `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。

フォルダー構造は以下のとおりです。

```
.
|— public-html
|   |— index.html
|— Dockerfile
|— taskdef.json
```

> **Note**
>
> 次の手順は、 CodeCatalyst 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が、必要に応じて Git を使用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クローニング」を参照してください。

トピック

- index.html
- Dockerfile
- taskdef.json

index.html

`index.html`このファイルには、Hello World メッセージがブラウザに表示されます。

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 [ファイルを作成] を選択します。
3. [ファイル名] に、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、public-html/必ずプレフィックスを含めてください。はこのフォルダにあるはずです。index.html

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.htmlpublic-htmlがリポジトリ内のフォルダーに追加されます。

Dockerfile

Dockerfile には、使用するベースの Docker イメージと、それに適用する Docker コマンドが記述されています。Dockerfile について詳しくは、『Dockerfile リファレンス』を参照してください。

ここで指定されている Dockerfile は、Apache 2.4 のベースイメージ () を使用するよう指示しています。`httpd`また、Web ページを提供する Apache `index.html` サーバー上のフォルダーに呼び出されるソースファイルをコピーする手順も含まれています。Dockerfile `EXPOSE` 内の命令は、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 [ファイルを作成] 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
   ```

   ファイル拡張子は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ーにある必要があります。`Docker build`ワークフローのコマンドは、そこにあることを想定しています。

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。

   Docker ファイルがリポジトリに追加されます。

taskdef.json

`taskdef.json`このステップで追加するファイルは、すでに指定したファイルと同じですが、次の点が異なります。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

このタスク定義では、image: field (httpd:2.4) にハードコードされた Docker イメージ名を指定する代わりに、2 つの変数 (および) を使用してイメージを表します。$REPOSITORY_URI $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 開発者ガイドの「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 [ファイルを作成] 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、次のように置き換えます。

arn: aws: iam:: #####ID: ###/-### codecatalyst-ecs-task-execution

でメモしたタスク実行ロールの ARN を使用します。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て Docker イメージにビルドし、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。

このワークフローは、順番に実行される以下の構成要素で構成されています。

- トリガー — このトリガーは、ソースリポジトリに変更を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、「トリガーの使用」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、イメージを Amazon ECR にプッシュします。また、taskdef.jsonimageビルドアクションはを正しいフィールド値で更新し、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、次に実行される deploy アクションの入力として使用されます。

  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でのワークフローを使用した構築 CodeCatalyst。
- 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探し、taskdef.jsonその内部を見つけて Amazon ECS サービスに登録します。その後、taskdef.jsonサービスはファイル内の指示に従って、Amazon ECS クラスター内で 3 つの Amazon ECS タスクとそれに関連する 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] には [] 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
```

前述のコードでは、以下を置き換えてください。

- *codecatalyst-ecs-environment*のインスタンスは両方とも、作成した環境の名前を使用してください。前提条件
- どちらのインスタンスも、*codecatalyst-account-connection*アカウント接続の表示名が付いています。表示名は数字の場合があります。詳細については、「ステップ 5: AWS ロールを追加 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と一緒にステップ 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、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
- *111122223333.dkr. ecr.us-west-2.amazonaws.com* (Run: aws ecrコマンド内) に Amazon ECR リポジトリの URI をイメージサフィックス () を除いたもの。/codecatalyst-ecs-image-repo
- *codecatalyst-ecs-deploy-role*で作成したデプロイロールの名前を使用。ステップ 4: AWS ロールを作成する
- *us-west-2* AWS の両方のインスタンスとリージョンコード 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

> **Note**
>
> ビルド & デプロイロールを作成しないことに決めた場合は、*codecatalyst-ecs-build-rolecodecatalyst-ecs-deploy-role*CodeCatalystWorkflowDevelopmentRole-*spaceName*とをロールの名前に置き換えてください。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**
>
> findsed前述のワークフローコードで示したおよび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、Render Amazon ECS タスク定義アクションを使用して

更新できます。詳細については、「「Amazon ECS タスク定義をレンダリング」アクションの追加」を参照してください。

8. (オプション) コミットする前に [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。
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」には、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] には、を選択しますcodecatalyst-ecs-source-repository。

    c. [ブランチ名] には [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。ワークフローの上部に定義されているトリガーにより、ワークフローの実行が自動的に開始されます。具体的には、workflow.yamlソースリポジトリにファイルをコミット (およびプッシュ) すると、トリガーによってワークフローの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. を選択してBuildBackend、ビルドの進行状況を確認します。
4. DeployToECS を選択してデプロイの進行状況を確認します。

実行詳細の表示に関する詳細については、を参照してくださいワークフロー実行のステータスと詳細の表示。

デプロイメントを確認するには

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。

   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す「Hello World」ページがブラウザに表示されます。

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、`index.html`ソースリポジトリ内のファイルに変更を加えます。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、Amazon ECR にプッシュして 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]、[ソースリポジトリ] の順に選択し、**`codecatalyst-ecs-source-repository`**リポジトリを選択します。
2. [`public-html`] を選択し、[`index.html`] を選択します。

   `index.html`の内容が表示されます。
3. [編集] を選択します。
4. 14 `Hello World Tutorial complete!` 行目のテキストをに変更します。
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て [コミットを表示] 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見る:
   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`codecatalyst-ecs-workflow`、最新の実行結果を表示します。
   c. と DeployToECS を選択してBuildBackend、ワークフローの実行の進行状況を確認します。
8. 以下のように、アプリケーションが更新されたことを確認します。
   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てください`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 Amazon ECR コンソールに切り替え AWS、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

課金されないように、このチュートリアルで使用しているファイルとサービスをクリーンアップしてください。

では AWS Management Console、次の順序でクリーンアップします。

1. Amazon ECS では、次の操作を行います。
   a. 削除`codecatalyst-ecs-service`。
   b. [削除] `codecatalyst-ecs-cluster`。
   c. `codecatalyst-ecs-task-definition` の登録を解除します。
2. Amazon ECR では、削除してください`codecatalyst-ecs-image-repo`。
3. Amazon EC2 では、`codecatalyst-ecs-security-group`削除します。
4. IAM ID センターで、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. 削除`codecatalyst-ecs-workflow`。
2. [削除] `codecatalyst-ecs-environment`。
3. [削除] `codecatalyst-ecs-source-repository`。
4. [削除] `codecatalyst-ecs-project`。

このチュートリアルでは、 CodeCatalyst ワークフローと Deploy to Amazon ECS 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びました。

チュートリアル:Amazon EKS へのアプリケーションのデプロイ

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 サービスにデプロイする方法を学びます。 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、必要な準備作業について説明し、次にアプリケーションをビルドしてクラスターにデプロイするためのワークフローを作成する方法について説明します。

初期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されます。この変更により、新しい Docker イメージがビルドされ、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、Amazon EKS の完全なセットアップを行う設計図を使用することもできます。EKS アプリケーションデプロイブループリントを使用する必要があります。詳細については、「ブループリントを使ったプロジェクトの作成」を参照してください。

トピック

- クリーンアップ

前提条件

このチュートリアルを始める前に:

- AWS アカウントが接続されている Amazon CodeCatalyst スペースが必要です。詳細については、「AWS Builder ID ユーザーをサポートするスペースの作成」を参照してください。
- スペースには、以下の名前の空の「ゼロから開始」 CodeCatalyst プロジェクトが必要です。

  ```
  codecatalyst-eks-project
  ```

  詳細については、「Amazon で空のプロジェクトを作成する CodeCatalyst」を参照してください。
- プロジェクトには、 CodeCatalyst 以下の名前の空のソースリポジトリが必要です。

  ```
  codecatalyst-eks-source-repository
  ```

  詳細については、「のソースリポジトリ CodeCatalyst」を参照してください。
- プロジェクトには、(開発環境ではなく) 次の名前の CodeCatalyst CI/CD 環境が必要です。

  ```
  codecatalyst-eks-environment
  ```

  この環境を以下のように設定します。
  - 「非実稼働」など、どのタイプでも選択できます。
  - AWS アカウントをそれConnect。

  詳細については、「環境を使用する」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で開発マシンを構成することです。これらのツールは以下のとおりです。

- `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 — の前提条件でもある `eksctl`

これらのツールは、既存の開発マシンがあればそのマシンにインストールすることも、クラウドベースの CodeCatalyst Dev Environment を使用することもできます。 CodeCatalyst Dev Environment の利点は、スピンアップや削除が容易で、 CodeCatalyst 他の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手順で進めることができることです。

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。

以下の説明では、 CodeCatalyst 開発環境を簡単に起動して必要なツールで設定する方法を説明していますが、詳細な手順が必要な場合は、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
- eksctl のインストールまたはアップグレード (『Amazon EKS ユーザーガイド』)
- 『AWS Command Line Interface ユーザーガイド』の最新バージョンのインストールまたは更新。 AWS CLI

開発環境を起動するには

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. 上部にある [開発環境を作成] を選択し、[AWS Cloud9 (ブラウザで)] を選択します。
6. 「既存のブランチで動作」と「メイン」が選択されていることを確認し、「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) がそのタブに複製されます。

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ることを確認します。

   kubectlインストールが完了しました。

eksctl をインストールして設定するには

> **Note**
>
> eksctlkubectl代わりにを使用できるので、厳密には必須ではありません。ただし、eksctlクラスター設定の多くを自動化できるという利点があるため、このチュートリアルではこのツールをお勧めします。

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ることを確認します。

   eksctlインストールが完了しました。

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. バージョンが表示されていることを確認して、 AWS CLI がインストールされていることを確認します。

   残りの手順を実行して、 AWS CLI AWSに必要なアクセス権限を設定してください。

を設定するには AWS CLI

AWS サービスにアクセスできるようにするには、アクセスキーとセッショントークンを使用してを設定する必要があります。 AWS CLI 以下の手順ではキーとトークンを簡単に設定できますが、詳細な手順が必要な場合は、『AWS Command Line Interface ユーザーガイド』 AWS CLIの「の設定」を参照してください。

1. 次のように IAM ID センターのユーザーを作成します。

   a. AWS Management Console にサインインし、https://console.aws.amazon.com/singlesignon/ AWS IAM Identity Center の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、[有効化] を選択する必要がある場合があります)。

> **Note**
>
> 必ず、 AWS アカウント CodeCatalystスペースに接続されているを使用してサインインしてください。どのアカウントが接続されているかは、スペースに移動して AWS アカウントタブを選択することで確認できます。詳細については、「AWS Builder ID ユーザーをサポートするスペースの作成」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] に、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を生成する」を選択します。

e. [メールアドレス] と [メールアドレスの確認] に、IAM Identity Center にまだ存在しないメールアドレスを入力します。

f. [名前] に、次のように入力します。

```
codecatalyst-eks-user
```

g. [姓] に、次のように入力します。

```
codecatalyst-eks-user
```

h. [表示名] には、次の内容を入力します。

```
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、「ユーザーを追加」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. [Copy] を選択し、ログ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、および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. 権限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. [定義済みの権限セット] を選択し、を選択しますAdministratorAccess。 AWS のサービスこのポリシーはすべてのユーザーに完全な権限を付与します。

   c. [次へ] をクリックします。

   d. [権限セット名] で、AdministratorAccess削除して次のように入力します。

      ```
      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. codecatalyst-eks-user権限セットを次のようにに割り当てます。

   a. ナビゲーションペインでを選択しAWS アカウント、 AWS アカウント 現在ログインしているアカウントの横にあるチェックボックスを選択します。

   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-usercodecatalyst-eks-permission-setとをに割り当てて AWS アカウント、まとめることができました。

4. codecatalyst-eks-userのアクセスキーとセッショントークンを次のように取得します。

   a. AWS のアクセスポータル URL、ユーザ名、ワンタイムパスワードがわかっていることを確認してください。codecatalyst-eks-userこの情報は事前にテキストエディタにコピーしておくべきでした。

> **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、[パスワードをリセット]、[ワンタイムパスワードの生成] を選択します [...] 、パスワードをもう一度リセットすると、情報が画面に表示されます。

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. 次の方法でログインします。

- ユーザー名:

```
codecatalyst-eks-user
```

- パスワード:

one-time-password

e. [新しいパスワードの設定] に新しいパスワードを入力し、[新しいパスワードを設定] 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、 AWS アカウント codecatalyst-eks-userユーザと権限セットを割り当てたの名前を選択します。

g. 次にcodecatalyst-eks-permission-set、[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。

h. ページの中央にあるコマンドをコピーします。これらは以下のようになります。

```
export AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
export AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
export AWS_SESSION_TOKEN="session-token"
```

... ここで、*##########################*。

5. 次のように、アクセスキーとセッショントークンをに追加します。 AWS CLI

a. CodeCatalyst 開発環境に戻ってください。

b. ターミナルプロンプトに、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の設定が完了しました。 AWS CLI これで、AWS CLI を使用してこのチュートリアルに必要なタスクを完了できます。

> **Important**
>
> このチュートリアルの実行中に、次のようなメッセージが表示されることがあります。
> `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たからです。この場合は、*`aws configure`*コマンドを実行しないでください。代わりに、この手順のステップ 4 `Obtain codecatalyst-eks-user's access key and session token` で始まる手順に従って、セッションを更新してください。

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。以下の説明では`eksctl`、を使用してクラスターを簡単に作成する方法を説明していますが、詳細な手順が必要な場合は、以下を参照してください。

- 『Amazon EKS ユーザーガイド』の eksctl の使用を開始する

  または

- コンソールと AWS CLIAmazon EKS ユーザーガイドの使用開始 (`kubectl`このトピックではクラスターの作成手順を説明します)

> **Note**
>
> プライベートクラスターは Amazon EKS CodeCatalyst との統合ではサポートされていません。

開始する前に

開発マシンで以下のタスクを完了していることを確認してください。

- eksctlユーティリティをインストールした。
- kubectlユーティリティをインストールした。
- AWS CLI をインストールし、アクセスキーとセッショントークンを設定した。

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**
>
> クラスターは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。以下のステップで説明されているように、eksctlユーティリティを使用してください。

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たい名前に置き換えられます。
   - *us-west-2 ##################*。

   10 ~ 20 分後に、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**
   >
   > AWS クラスターを作成すると、waiting for CloudFormation stack複数の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 に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 に空のリポジトリを作成します。

```
aws ecr create-repository --repository-name codecatalyst-eks-image-repo
```

Amazon ECR *codecatalyst-eks-image-repo*リポジトリに付けたい名前に置き換えます。

このチュートリアルでは、リポジトリに名前を付けたことを前提としています。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 (例:) を書き留めます。111122223333.dkr.ecr.us-west-2.amazonaws.com/codecatalyst-eks-image-repo

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、アプリケーションソースファイルをソースリポジトリ (codecatalyst-eks-source-repository) に追加します。これらには以下のものが含まれます。

- index.htmlファイル —「ハロー、ワールド！」と表示されます。ブラウザにメッセージが表示される。
- Dockerファイル — 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 コマンドを記述します。
- deployment.yamlファイル — Kubernetes サービスとデプロイメントを定義する Kubernetes マニフェスト。

フォルダ構造は以下のとおりです。

```
|– 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。ブラウザにメッセージが表示される。

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、public-htmlというフォルダを作成します。
3. で/public-html、index.html以下の内容を含む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`リポジトリ内のフォルダに追加されます。

Dockerfile

Dockerfile には、使用するベースの Docker イメージと、それに適用する Docker コマンドが記述されています。Dockerfile について詳しくは、『Dockerfile リファレンス』を参照してください。

ここで指定されている Dockerfile は、Apache 2.4 のベースイメージ () を使用するよう指示しています。`httpd`また、Web ページを提供する Apache `index.html` サーバー上のフォルダーに呼び出されるソースファイルをコピーする手順も含まれています。Dockerfile `EXPOSE` 内の命令は、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、Dockerfile以下の内容を含む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
```

ファイル拡張子は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ーにある必要があります。Docker buildワークフローのコマンドは、そこにあることを想定しています。

2. 追加、コミット、プッシュ:

```
git add .
git commit -m "add Dockerfile"
git push
```

Docker ファイルがリポジトリに追加されます。

deployment.yaml

このセクションでは、deployment.yamlリポジトリにファイルを追加します。deployment.yamlこのファイルは Kubernetes マニフェストで、実行する 2 種類の Kubernetes リソースタイプまたは種類を定義しています。1 つは「サービス」と「デプロイ」です。

- 「サービス」は Amazon EC2 にロードバランサーをデプロイします。ロードバランサーはインターネット向けのパブリック URL と標準ポート (ポート 80) を提供し、これを使用して「Hello, World!」をブラウズできます。アプリケーションをデプロイします。
- 「デプロイ」では 3 つのポッドがデプロイされ、各ポッドには「Hello, World!」と書かれた 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、クラスターを作成したとき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、Kubernetes リソースタイプをいくつでも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。

デプロイメント.yaml ファイルを追加するには

1. でcodecatalyst-eks-source-repository、というフォルダを作成します。Kubernetes
2. で/Kubernetes、deployment.yaml以下の内容を含む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このファイルは、リポジトリのという名前のフォルダーに追加されますKubernetes。

これで、すべてのソースファイルが追加されました。

作業内容を再確認し、すべてのファイルが正しいフォルダに配置されていることを確認してください。フォルダー構造は以下のとおりです。

```
|— 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。これらのロールは以下のとおりです。

- ビルドロール — AWS アカウントにアクセスして Amazon ECR と Amazon EC2 に書き込むためのアクセス権限を (ワークフロー内の) CodeCatalyst ビルドアクションに付与します。
- デプロイロール — Kubernetes CodeCatalyst へのデプロイクラスターアクション (ワークフロー内) に、 AWS アカウントと Amazon EKS にアクセスする権限を付与します。

IAM ロールの詳細については、ユーザーガイドの「IAM ロール」を参照してください。AWS Identity and Access Management

> **Note**
>
> 時間を節約するために、前述の 2 つのロールの代わりに、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用の CodeCatalystWorkflowDevelopmentRole-***spaceName*** ロールの作成」を参照してください。CodeCatalystWorkflowDevelopmentRole-*spaceName*このロールには非常に幅広い権限があり、セキュリティ上のリスクが生じる可能性があることを理解してください。このロールは、セキュリティがそれほど問題になら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*このディレクトリに、codecatalyst-eks-trust-policy.json以下の内容の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*ディレクトリに、codecatalyst-eks-build-policy.json以下の内容を含むというファイルを作成します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:*",
                "ec2:*"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> 初めてロールを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、リソースが使用可能になったら、リソース名を使用して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*ディレクトリに、codecatalyst-eks-deploy-policy.json以下の内容を含むというファイルを作成します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeCluster",
```

```
                "eks:ListClusters"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> 初めてロールを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、リソースが使用可能になったら、リソース名を使用して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。これで、ディレクトリ構造は以下のようになりました。

```
|— 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. "arn":コマンド出力の値 (例:) arn:aws:iam::111122223333:policy/codecatalyst-eks-build-policy を書き留めます。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、"arn":デプロイポリシーの値 (例:) arn:aws:iam::111122223333:policy/codecatalyst-eks-deploy-policy を書き留めます。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、先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. "Arn":ロールの値 (例:) arn:aws:iam::111122223333:role/codecatalyst-eks-build-role を書き留めます。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、先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. "Arn":ロールの値 (例:) arn:aws:iam::111122223333:role/codecatalyst-eks-deploy-role を書き留めます。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 ARN を記録しました。

ステップ 6: AWS ロールをに追加 CodeCatalyst

このステップでは、スペースに接続したものにビルドロール (codecatalyst-eks-build-rolecodecatalyst-eks-deploy-role) とデプロイロール () を追加します。 AWS アカウント 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自分に追加するには 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. 上部の [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 表示名列に、 AWS アカウント ビルドロールとデプロイロールを作成した場所の表示名をコピーします。(数字の場合もあります)。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. 管理コンソールから [AWS ロールを管理] を選択します。

   「Amazon CodeCatalyst スペースに IAM ロールを追加」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。リストには、ビルドロールとデプロイロール、codecatalyst-runner.amazonaws.comcodecatalyst.amazonaws.comおよびサービスプリンシパルを含む信頼ポリシーが設定されたその他の 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**
>
> 表示される場合はThe security token included in the request is invalid、適切な権限がないことが原因である可能性があります。この問題を解決するには、サインアウトして、 AWS CodeCatalystスペースを作成したときに使用したアカウントでサインインし直してください。 AWS

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るはずです。

   これで、 CodeCatalyst これらのロールを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

作成したデプロイロールを Kubernetes ステップ 5: AWS ロールを作成する ConfigMap ファイルに追加して、(ワークフロー内の) Deploy to Kubernetes クラスターアクションが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またはを使用してこのタスクを実行できます。kubectl

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ことに決めた場合は、*codecatalyst-eks-deploy-role*CodeCatalystWorkflowDevelopmentRole-*spaceName*そのロールの名前に置き換えてください。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 ###################*。

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. 赤色のイタリック体のテキストを追加します。

```
# 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ことに決めた場合は、*codecatalyst-eks-deploy-role*CodeCatalystWorkflowDevelopmentRole-*spaceName*そのロールの名前に置き換えてください。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、ひいては Deploy to Amazon EKS アクションに Kubernetes system:masters クラスターへのアクセス権限が付与されました。

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て Docker イメージにビルドし、そのイメージを Amazon EKS クラスターのツリーポッドに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、順番に実行される以下のビルディングブロックで構成されています。

- トリガー — このトリガーは、ソースリポジトリに変更を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、「トリガーの使用」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、イメージを Amazon ECR にプッシュします。また、$REPOSITORY_URI$IMAGE_TAGdeployment.yamlビルドアクションはファイル内のおよび

変数を正しい値で更新し、このファイルとフォルダ内の他のファイルの出力アーティファクトを作成します。Kubernetesこのチュートリアルでは、Kubernetesdeployment.yamlフォルダー内の唯一のファイルはですが、さらにファイルを含めることもできます。このアーティファクトは、次のデプロイアクションの入力として使用されます。

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でのワークフローを使用した構築CodeCatalyst。

- 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探し、deployment.yamlその中のファイルを検索します。その後、deployment.yamlアクションはファイル内の指示に従って 3 つのポッドを実行します。各ポッドには「Hello, World!」が 1 つ入っています。ドッカーコンテナ — Amazon EKS クラスター内にあります。

ワークフローを作成するには

1. コンソールに移動します。 CodeCatalyst
2. プロジェクト (codecatalyst-eks-project) に移動します。
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。
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] には [] 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定義リファレンス。

```
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/
```

前述のコードでは、以下を置き換えてください。

- *codecatalyst-eks-environment*のインスタンスは両方とも、作成した環境の名前を使用してください。前提条件
- どちらのインスタンスも、*codecatalyst-account-connection*アカウント接続の表示名が付いています。表示名は数字の場合があります。詳細については、「ステップ 6: AWS ロールをに追加 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と一緒にステップ 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、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
- *111122223333.dkr. ecr.us-west-2.amazonaws.com* (Run: aws ecrコマンド内) に Amazon ECR リポジトリの URI をイメージサフィックス () を除いたもの。/ codecatalyst-eks-image-repo
- *codecatalyst-eks-deploy-role*で作成したデプロイロールの名前で。ステップ 5: AWS ロールを作成する

- *us-west-2* AWS の両方のインスタンスとリージョンコード。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

> **Note**
>
> ビルド & デプロイロールを作成しないことに決めた場合は、*codecatalyst-eks-build-role*`codecatalyst-eks-deploy-role`CodeCatalystWorkflowDevelopmentRole-*spaceName*とをロールの名前に置き換えてください。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) コミットする前に [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. [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」には、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] には、を選択しますcodecatalyst-eks-source-repository。

    c. [ブランチ名] には [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。ワークフローの上部に定義されているトリガーにより、ワークフローの実行が自動的に開始されます。具体的には、workflow.yamlソースリポジトリにファイルをコミット (およびプッシュ) すると、トリガーによってワークフローの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. 作成したワークフローを選択します。codecatalyst-eks-workflow

3. を選択してBuildBackend、ビルドの進行状況を確認します。
4. DeployToEKS を選択すると、デプロイの進行状況が表示されます。

   実行詳細の表示に関する詳細は、を参照してくださいワークフロー実行のステータスと詳細の表示。

デプロイを検証するには

1. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
2. 左側の下部にある [ロードバランサー] を選択します。
3. Kubernetes デプロイメント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ればよいかわからない場合は、「タグ」タブで以下のタグを探してください。
   - `kubernetes.io/service-name`
   - `kubernetes.io/cluster/ekstutorialcluster`
4. 正しいロードバランサーを選択したら、[Description] 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。

   「ハロー、ワールド！」 Web 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が示されます。

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、`index.html`ソースリポジトリ内のファイルに変更を加えます。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、Amazon ECR にプッシュして 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、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てください。

```
git pull
```

4. codecatalyst-eks-source-repository/public-html/index.html を開きます。
5. 14 Hello, World! Tutorial complete! 行目のテキストをに変更します。
6. 追加、コミット、プッシュ:

```
git add .
git commit -m "update index.html title"
git push
```

ワークフローの実行が自動的に開始されます。

7. (オプション) 以下を入力します。

```
git show HEAD
```

index.html変更のコミット ID をメモしておきます。このコミット ID は、開始したばかりの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見る:
   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. codecatalyst-eks-workflow最新の実行結果を表示するように選択します。
   c. と DeployToEKS を選択するとBuildBackend、ワークフローの実行の進行状況が表示されます。
9. 以下のように、アプリケーションが更新されたことを確認します。
   a. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
   b. 左側の下部にある [ロードバランサー] を選択します。
   c. Kubernetes デプロイメント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。

      「チュートリアル完了！」 Web 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が示されます。
10. (オプション) で Amazon ECR コンソールに切り替え AWS、新しい Docker イメージにこの手順のステップ 7 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したストレージとコンピューティングリソースに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:

   - 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 ##################*。
     - *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 分後に、クラスターと関連リソースが削除されます。これには、 AWS CloudFormation スタック、ノードグループ (Amazon EC2 内)、ロードバランサーなどが含まれますが、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 認証情報または認証情報を更新する必要がある場合があります。`kubectl`どの認証情報を更新すればよいかわからない場合は、 AWS まず認証情報を更新してください。 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」エラーと「ExpiredToken」エラーを修正する方法を教えてください。。`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方法を教えてください。。

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 では、削除してください`codecatalyst-eks-image-repo`。
   2. IAM ID センターで、以下を削除します。

      a. `codecatalyst-eks-user`

b. codecatalyst-eks-permission-set

3. IAM では、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. 削除codecatalyst-eks-workflow。
2. 削除codecatalyst-eks-environment。
3. 削除codecatalyst-eks-source-repository。
4. 開発環境を削除します。
5. 削除codecatalyst-eks-project。

このチュートリアルでは、 CodeCatalyst ワークフローと Deploy to Kubernetes クラスター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びました。

「 AWS CloudFormation デプロイスタック」アクションの追加

> **Tip**
>
> Deploy AWS CloudFormation Stack 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: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。

このセクションでは、Deploy AWS CloudFormation stack アクションをワークフローに追加する方法について説明します。このアクションは、 CloudFormation AWS 指定したテンプレートに基づいてスタックを作成します。テンプレートには次のようなものがあります。

- AWS CloudFormation テンプレート — 詳細については、「AWS CloudFormation テンプレートの使用」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、 AWS SAM `sam package`まずオペレーションを使用して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ングを自動的に行う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、 CloudFormation `CreateChangeSet`アクションはオペレーションを実行し、`ExecuteChangeSet`次に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「成功」または「失敗」のマークを付けます。

AWS CloudFormation AWS SAM デプロイしたいリソースを含むテンプレートがすでにある場合や、AWS SAM やなどのツールを使用してワークフロー構築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、Deploy AWS CloudFormation スタックアクションを使用してください。AWS Cloud Development Kit (AWS CDK)

CloudFormation 作成できるテンプレートや AWS SAM Deploy AWS CloudFormation Stack アクションで使用できるテンプレートに制限はありません。

Visual

ビジュアルエディターを使用して「Deploy AWS CloudFormation stack」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 [Amazon] を選択します CodeCatalyst。

9. Deploy AWS CloudFormation スタック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は:
     - (オプション)「ダウンロード」を選択すると、アクションのソースコードが表示されます。
     - 「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AWS CloudFormation 「スタックのデプロイ」アクションリファレンス。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. [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「 AWS CloudFormation スタックをデプロイ」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 [Amazon] を選択します CodeCatalyst。
9. Deploy AWS CloudFormation スタック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は:
     - (オプション)「ダウンロード」を選択すると、アクションのソースコードが表示されます。
     - 「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、AWS CloudFormation 「スタックのデプロイ」アクションリファレンス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。

「 AWS CloudFormation スタックをデプロイ」アクションによって生成される変数

Deploy AWS CloudFormation stack アクションを実行すると、後続のワークフローアクションで使用できる変数が生成されます。詳細については、「AWS CloudFormationデプロイスタック」アクション変数のを参照してください定義済み変数のリスト。

「 AWS CloudFormation デプロイスタック」アクション定義

Deploy AWS CloudFormation stack 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、AWS CloudFormation 「スタックのデプロイ」アクションリファレンスワークフロー定義リファレンスのを参照してください。

「Amazon ECS にデプロイ」アクションの追加

> **Tip**
>
> Deploy to Amazon ECS 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 へのアプリケーションのデプロイ。

> **Tip**
>
> Amazon ECS へのデプロイアクションの実際の例として、Node.js API とブループリントを使用するか、Java API AWS Fargate AWS Fargateとブループリントを使用してプロジェクトを作成します。詳細については、「ブループリントを使ったプロジェクトの作成」を参照してください。

このセクションでは、ワークフローに Deploy to Amazon ECS アクションを追加する方法について説明します。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。登録すると、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。「タスク定義のインスタンス化」は、Amazon ECS にアプリケーションを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、既存の Amazon ECS クラスター、サービス、およびタスク定義ファイルが用意されている必要があります。

Amazon ECS の詳細については、Amazon エラスティックコンテナサービス開発者ガイド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 にデプロイ」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 [Amazon] を選択します CodeCatalyst。
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

   - [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は:
     - (オプション)「ダウンロード」を選択すると、アクションのソースコードが表示されます。
     - 「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「Amazon ECS へのデプロイ」アクションリファレンス。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. [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 エディタを使用して「Amazon ECS にデプロイ」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 [Amazon] を選択します CodeCatalyst。
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

   - [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は:
     - (オプション)「ダウンロード」を選択すると、アクションのソースコードが表示されます。
     - 「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、「Amazon ECS へのデプロイ」アクションリファレンス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。

「Amazon ECS にデプロイ」アクションによって生成された変数

Deploy to Amazon ECS アクションを実行すると、後続のワークフローアクションで使用できる変数が生成されます。詳細については、「Amazon ECS にデプロイ」アクション変数の 定義済み変数のリスト「」を参照してください。

「Amazon ECS にデプロイ」アクション定義

Amazon ECS へのデプロイアクションは、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、「Amazon ECS へのデプロイ」アクションリファレンスのを参照してください。ワークフロー定義リファレンス

「Kubernetes クラスターへのデプロイ」アクションの追加

> **Tip**
>
> Deploy to Kubernetes クラスター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 へのアプリケーションのデプロイ

このセクションでは、Deploy to Kubernetes クラスターアクションをワークフローに追加する方法について説明します。このアクションは、お客様が提供する 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。マニフェストのサンプルについては、を参照してください。deployment.yaml チュートリアル:Amazon EKS へのアプリケーションのデプロイ

Kubernetes の詳細については、Kubernetes の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 とは何ですか?」を参照してください。 Amazon EKS ユーザーガイドに記載されています。

仕組み

Kubernetes へのデプロイクラスターは次のように機能します。

1. 実行時に、 CodeCatalystアクションはアクションが実行されているコンピュートマシンに 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、`kubectl`アクションを設定したときに指定した Amazon EKS クラスターを指すように設定されます。次に、`kubectl`kubectl applyコマンドを実行するにはユーティリティが必要です。
2. `kubectl apply -f` *`my-manifest.yaml`*アクションはコマンドを実行し、*`my-manifest.yaml`* 内の指示を実行して、アプリケーションをコンテナと Pod のセット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。

前提条件

このアクションを使用するには、以下の準備が必要です。

> **Tip**
>
> これらの前提条件をすばやく設定するには、の指示に従ってください。チュートリアル:Amazon EKS へのアプリケーションのデプロイ

- 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記述した Dockerfile が少なくとも 1 つ必要です。Dockerfile について詳しくは、Dockerfile リファレンスをご覧ください。
- 少なくとも 1 つの Kubernetes マニフェストファイル (Kubernetes ドキュメントでは設定ファイルまたは設定と呼ばれています)。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。
- Deploy to Kubernetes クラスターアクションに Amazon EKS クラスターへのアクセスと操作を許可する IAM ロールです。詳細については、「Kubernetes クラスターへのデプロイ」アクションリファレンス 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 ファイル ConfigMap 。 ConfigMap ファイルにロールを追加する方法については、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。
  - 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、 AWS アプリケーションをデプロイするアカウントに接続されている必要があります。詳細については、AWS Builder ID ユーザーをサポートするスペースの作成、Amazon で空のプロジェクトを作成する CodeCatalyst、および環境を使用するを参照してください。
- CodeCatalystがサポートするソースリポジトリ。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 マニフェストが格納されます。詳細については、「のソースリポジトリ CodeCatalyst」を参照してください。

「クラスターにデプロイ」アクションを追加します。

次の手順に従って、Deploy to cluster 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ーを使用して「Kubernetes クラスターにデプロイ」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 [Amazon] を選択します CodeCatalyst。
9. Kubernetes へのデプロイ (Deploy to Kubernetes) クラスター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、設定ペインを開きます。

     または

   - [Kubernetes クラスタにデプロイ] 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は:
     - (オプション)「ダウンロード」を選択すると、アクションのソースコードが表示されます。
     - 「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「Kubernetes クラスターへのデプロイ」アクションリファレンス。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. [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「Kubernetes クラスターにデプロイ」アクションを追加するには

1. https://codecatalyst.aws/ でコンソールを開きます。 CodeCatalyst
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 [Amazon] を選択します CodeCatalyst。
9. Kubernetes へのデプロイ (Deploy to Kubernetes) クラスター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、設定ペインを開きます。

     または

   - [Kubernetes クラスタにデプロイ] 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は:
     - (オプション)「ダウンロード」を選択すると、アクションのソースコードが表示されます。
     - 「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、「Kubernetes クラスターへのデプロイ」アクションリファレンス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。

「Kubernetes クラスタにデプロイ」アクションによって生成された変数

Deploy to Kubernetes クラスターアクションを実行すると、後続のワークフローアクションで使用できる変数が生成されます。詳細については、の「」を参照してください「Kubernetes クラスターにデプロイ」アクション変数。定義済み変数のリスト

「Kubernetes クラスタにデプロイ」アクション定義

Kubernetes へのデプロイクラスターアクションは、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、のを参照してください。「Kubernetes クラスターへのデプロイ」アクションリファレンス ワークフロー定義リファレンス

「AWS CDK デプロイ」アクションの追加

このセクションでは、AWS CDK デプロイアクションをワークフローに追加する方法について説明します。AWS CDK AWS Cloud Development Kit (AWS CDK) デプロイアクションはアプリを統合してにデプロイします。 AWSアプリケーションがすでにに存在する場合 AWS、アクションは必要に応じてアプリケーションを更新します。

を使用してアプリを作成する方法に関する一般的な情報については AWS CDK、「What is the AWS CDK?」を参照してください。 『AWS Cloud Development Kit (AWS CDK) 開発者ガイド』の。

トピック

- このアクションをいつ使うべきか
- 仕組み
- 「AWS CDK デプロイ」アクションで使用される CDK CLI バージョン
- アクションはいくつのスタックをデプロイできますか?
- 前提条件
- ワークフローの例
- 「deploy」AWS CDK アクションの追加
- 「AWS CDK deploy」アクションによって生成される変数
- 「AWS CDK デプロイ」アクション定義

このアクションをいつ使うべきか

このアクションは、を使用してアプリを開発し AWS CDK、自動継続的インテグレーションとデリバリー (CI/CD) ワークフローの一部として自動的にデプロイしたい場合に使用します。たとえば、アプリソースに関連するプルリクエストがマージされるたびに、 AWS CDK アプリを自動的にデプロイしたい場合があります。 AWS CDK

仕組み

AWS CDK デプロイの仕組みは以下のとおりです。

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 (AWS CDK ツールキットとも呼ばれる) をビルドイメージにダウンロードします。 CodeCatalyst

   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 にバンドルされているため、ダウンロードは行われません。
2. アクションは CDK CLI `cdk deploy` を使用してコマンドを実行します。このコマンドは、アプリを合成してにデプロイします。 AWS CDK AWSこのコマンドの詳細については、『開発者ガイド』の「AWS CDK Toolkit (cdk コマンド)」トピックを参照してください。AWS Cloud Development Kit (AWS CDK)

「AWS CDK デプロイ」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さまざまなバージョンでデフォルトで使用されるCDK CLI のバージョンを示しています。

> **Note**
>
> デフォルトは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」アクションリファレンス」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 またはそれ以降	2.99.1

アクションはいくつのスタックをデプロイできますか?

AWS CDK デプロイでは 1 つのスタックしかデプロイできません。 AWS CDK アプリケーション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含む親スタックを作成し、このアクションを使用して親スタックをデプロイする必要があります。

前提条件

AWS CDK deploy アクションを使用する前に、以下のタスクを完了してください。

1. AWS CDK アプリを用意してください。 AWS CDK v1 または v2 を使用して、 AWS CDK がサポートする任意のプログラミング言語でアプリケーションを作成できます。 AWS CDK AWS CDK アプリケーションファイルが次の場所で入手可能であることを確認してください。
   - CodeCatalyst ソースリポジトリ、または
   - CodeCatalyst 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。
2. 環境をブートストラップします。 AWS ブートストラップには以下の方法があります。
   - 『AWS Cloud Development Kit (AWS CDK) 開発者ガイド』の「ブートストラップの方法」で説明されている方法のいずれかを使用してください。
   - AWS CDK ブートストラップアクションを使用してください。このアクションは、AWS CDK デプロイと同じワークフローに追加することも、別のワークフローに追加することもできます。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を少なくとも 1 回実行して、必要なリソースを確保してください。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。「AWS CDK ブートストラップ」アクションの追加

     ブートストラップについて詳しくは、『開発者ガイド』の「ブートストラップ」を参照してください。AWS Cloud Development Kit (AWS CDK)

ワークフローの例

以下のワークフロー例には、AWS CDK AWS CDK デプロイアクションとブートストラップアクションが含まれています。このワークフローは、順番に実行される以下の構成要素で構成されています。

> **Note**
>
> 以下のワークフロー例は説明のためのもので、追加の設定がないと動作しません。これは、AWS CDK デプロイアクションで構成されたワークフローがどのようになるかを示すためのものです。

- トリガー — このトリガーは、ソースリポジトリに変更を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。 AWS CDK このリポジトリにはアプリケーションが含まれます。トリガーについての詳細は、「トリガーの使用」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) — トリガーされると、CDKToolkitアクションはブートストラップスタックをにデプロイします。 AWSCDKToolkitスタックが環境にすでに存在する場合は、必要に応じてアップグレードされます。それ以外の場合は何も起こらず、アクションは成功とマークされます。
- 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 AWS CDK デプロイアクションはアプリコードをテンプレートに統合し、AWS CloudFormation 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
```

「deploy」AWS CDK アクションの追加

次の手順に従って、AWS CDK デプロイ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、で説明されているタスクを完了していることを確認してください前提条件。

Visual

ビジュアルエディターを使用して「AWS CDK deploy」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 [Amazon] を選択します CodeCatalyst。
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

- [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は:
  - (オプション)「ダウンロード」を選択すると、アクションのソースコードが表示されます。
  - 「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「AWS CDK デプロイ」アクションリファレンス。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. [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。

> **Note**
>
> AWS CDK **`npm install`**デプロイアクションがエラーで失敗した場合、「npm install」エラーを修正するにはどうすればいいですか?エラーの修正方法については を参照してください。

YAML

YAML エディタを使用して「AWS CDK deploy」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 [Amazon] を選択します CodeCatalyst。
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

   - [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は:
     - (オプション)「ダウンロード」を選択すると、アクションのソースコードが表示されます。
     - 「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、「AWS CDK デプロイ」アクションリファレンス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。

> **Note**
>
> AWS CDK **`npm install`**デプロイアクションがエラーで失敗した場合、「npm install」エラーを修正するにはどうすればいいですか?エラーの修正方法については を参照してください。

「AWS CDK deploy」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を実行すると、後続のワークフローアクションで使用できる変数が生成されます。詳細については、AWS CDKアクション変数を「デプロイ」します。のを参照してください定義済み変数のリスト。

「AWS CDK デプロイ」アクション定義

AWS CDK デプロイアクションは、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、「AWS CDK デプロイ」アクションリファレンスワークフロー定義リファレンスのを参照してください。

デプロイでの作業

このセクションでは、Amazon CodeCatalyst でのデプロイを管理、監視、設定する方法について説明します。

トピック

- アプリケーションまたはリソースのデプロイ
- デプロイを再実行する
- デプロイステータス、コミット、プルリクエストを表示する
- デプロイログを表示する
- ロールバックの設定
- デプロイされたアプリケーションの URL を表示する
- デプロイメントターゲットの削除

アプリケーションまたはリソースのデプロイ

Amazon を使用し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、CodeCatalyst ワークフローを使用する必要があります。このワークフローには、何かをデプロイできるアクションが少なくとも 1 つ含まれている必要があります。たとえば、 AWS CloudFormation スタックをデプロイする場合は、Deploy AWS CloudFormation stack アクションを含むワークフローを作成します。

ワークフローの詳細については、「のワークフローによるビルド、テスト、デプロイ CodeCatalyst」を参照してください。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の使用。

デプロイを再実行する

Amazon でデプロイを再実行するには CodeCatalyst、関連するワークフローを再実行します。手順については、「ワークフロー実行の開始」を参照してください。

デプロイステータス、コミット、プルリクエストを表示する

Amazon のデプロイに関する次の情報を表示できます CodeCatalyst。

- デプロイアクティビティ (デプロイステータス、開始時間、終了時間、履歴、イベントの期間など)。
- スタック名 AWS リージョン、最終更新日時、および関連するワークフロー。
- コミットとプルリクエスト。
- アクション固有の情報 (CloudFormation イベントや出力など)。

ワークフロー、環境、またはワークフローアクションから始まるデプロイ情報を表示できます。

ワークフローから始ま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ランに移動します。手順については、「ワークフロー実行のステータスと詳細の表示」を参照してください。

環境から始まるデプロイ情報を表示するには

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、[環境] を選択します。
4. スタックがデプロイされた環境 (例:) を選択します。`Production`
5. Deployment activity を選択すると、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が表示されます。
6. Deployment target を選択すると、環境にデプロイされたスタック、クラスタ、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。

アクションから始まるデプロイ情報を表示するには

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. 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。たとえば、DeployCloudFormationStackを選択することができます。
6. 右側のペインの内容で、アクション固有のデプロイ情報を確認してください。

デプロイログを表示する

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 CodeCatalyst の問題をトラブルシューティングできます。

ワークフローまたは環境から始まるログを表示できます。

ワークフローから開始されるデプロイアクションのログを表示するには

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. [Runs] 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ランを選択します。
7. ワークフローダイアグラムで、ログを表示したい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. さらにログを表示するには、[Summary] タブを選択し、[View in] CloudFormation (表示可能な場合) を選択するとさらにログが表示されます。へのサインインが必要な場合があります AWS。

環境から開始されたデプロイアクションのログを表示するには

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、[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、Workflow Run ID 列を見つけ、スタックをデプロイしたワークフローランを選択します。

6. ワークフローダイアグラムで、ログを表示したい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. さらにログを表示するには、[Summary] タブを選択し、[View in] CloudFormation (表示可能な場合) を選択するとさらにログが表示されます。へのサインインが必要な場合があります AWS。

ロールバックの設定

デフォルトでは、Deploy AWS CloudFormation stack アクションが失敗した場合、スタックは最後に確認された安定した状態にロールバックされます。 AWS CloudFormation アクションが失敗したときだけでなく、指定した 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 CloudWatch 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。

CloudFormation アクションが失敗してもスタックがロールバックされないように、デフォルトの動作を変更することもできます。

次の手順に従ってロールバックを設定します。

> **Note**
>
> ロールバックは手動で開始できません。

Visual

開始する前に

1. 機能する Deploy AWS CloudFormation Stack 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、「「 AWS CloudFormation デプロイスタック」アクションの追加」を参照してください。
2. Deploy AWS CloudFormation スタックアクションの Stack role- options フィールドで指定されているロールには、CloudWatchFullAccess必ず権限を含めてください。適切な権限を使用してこのロールを作成する方法については、を参照してくださいステップ 2: AWS ロールを作成する。

「Deploy AWS CloudFormation stack」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/ 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. Deploy AWS CloudFormation スタック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 デプロイスタック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. 下部にある [詳細設定] を展開します。
10. [アラーム ARN の監視] で [アラームを追加] を選択します。
11. 次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。ロールバックトリガーは最大 5 つまで設定できます。

      > **Note**
      >
      > CloudWatch アラーム ARN を指定する場合、 CloudWatchアクションがアクセスできるようにするための追加の権限も設定する必要があります。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

      CloudFormation 指定したアラームを監視する時間を 0 ~ 180 分の間で指定します。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。指定したモニタリング時間内にアラームが発生すると、デプロイは失敗し、 CloudFormation スタック操作全体がロールバックされます。

      デフォルト:0。 CloudFormation スタックリソースがデプロイされている間だけアラームを監視し、デプロイ後は監視しません。

YAML

「Deploy AWS CloudFormation stack」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/ 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. Deploy AWS CloudFormation スタック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 `monitor-alarm-arns` `monitor-timeout-in-minutes` コードにおよび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」アクションリファレンス
8. Deploy AWS CloudFormation stack **`role-arn`** アクションのプロパティで指定されているロールには、CloudWatchFullAccess必ず権限を含めてください。適切な権限でこのロールを作成する方法については、を参照してくださいステップ 2: AWS ロールを作成する。

Visual

「Deploy AWS CloudFormation stack」アクションのロールバックを無効にするには

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. Deploy AWS CloudFormation スタック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 デプロイスタック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. 下部にある [詳細設定] を展開します。

10. [ロールバックを無効にする] をオンにします。

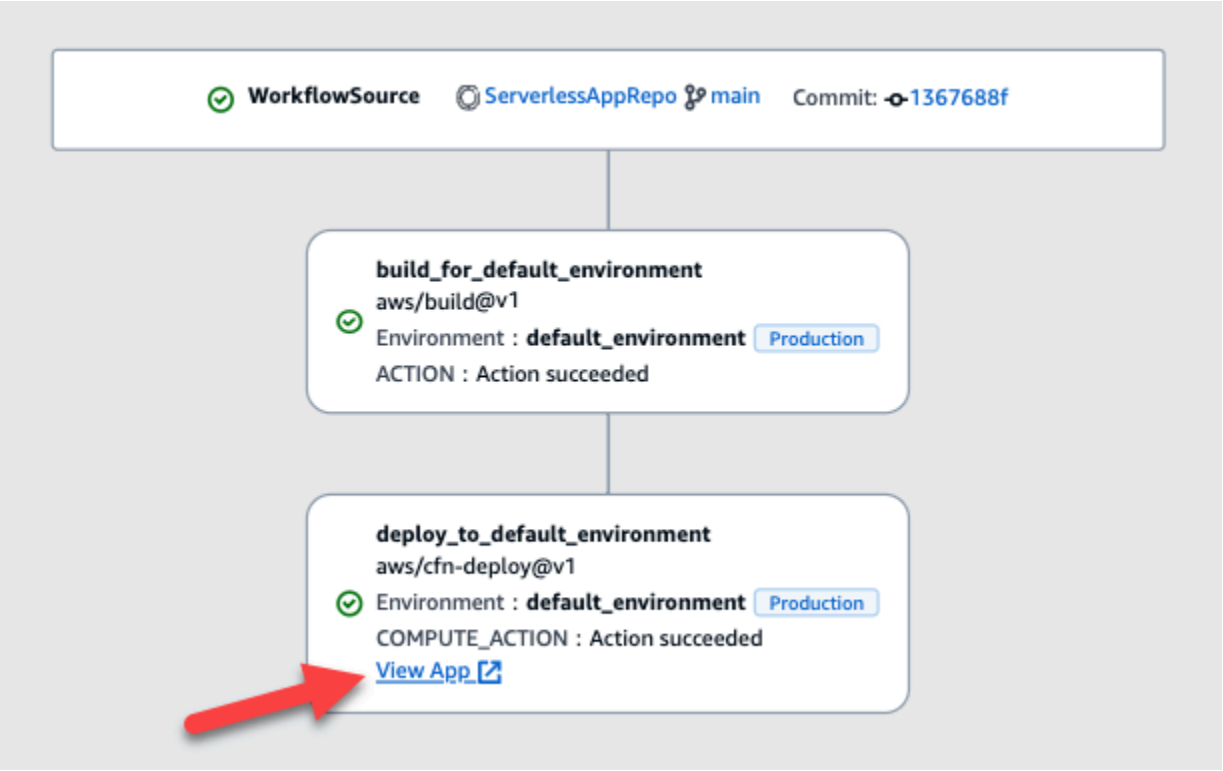
YAML

「Deploy AWS CloudFormation stack」アクションのロールバックをオフにするには

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. Deploy AWS CloudFormation スタック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 `disable-rollback: 1` コードにプロパティを追加してロールバックを停止します。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」アクションリファレンス

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 CodeCatalyst をクリック可能なリンクとして表示するように Amazon を設定できます。このリンクは、 CodeCatalyst デプロイしたアクション内のコンソールに表示されます。次のワークフロー図は、アクションの下部に表示される View App URL を示しています。



この URL CodeCatalyst を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迅速に検証できます。

> **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`、`endpointurl`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。名前はダッシュ (-)、アンダースコア ()、スペース (_) の有無にかかわらず使用できます。文字列は大文字と小文字を区別しません。変数の値を、`http`デプロイしたアプリケーションのまたは `https` URL に設定します。

> **Note**
>
> `app url`、`endpoint url`または文字列を含むように既存の出力変数を更新する場合は、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するようにします。

詳細な手順については、以下の手順のいずれかを参照してください。

- 「AWS CDK deploy」アクションでアプリ URL を表示するには
- 「 AWS CloudFormation デプロイスタック」アクションにアプリの URL を表示するには
- 他のすべてのアクションでアプリの URL を表示するには

URL の設定が完了したら、以下の手順に従って 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

「AWS CDK deploy」アクションでアプリ URL を表示するには

1. AWS CDK deploy アクションを使用している場合は、CfnOutput AWS CDK アプリケーションコードにコンストラクト (キーと値のペア) を追加します。

   - キー名にはappurl、またはendpointurl、結合ダッシュ (-)、アンダースコア ()、またはスペース (_) を含める必要があります。 文字列は大文字と小文字を区別しません。
   - 値は、httpデプロイされたアプリケーションのまたは https URL でなければなりません。

   たとえば、 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 CfnOutputProps リファレンスの「interface」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 デプロイスタック」アクションにアプリの URL を表示するには

1. Deploy AWS CloudFormation stack アクションを使用している場合は、Outputs CloudFormation AWS SAM テンプレートまたはテンプレートのセクションに次のような特徴を持つ出力を追加します。

   - キー (論理 ID とも呼ばれる) にはappurl、またはendpointurl、ダッシュ (-)、アンダースコア ()、またはスペース (_) を含める必要があります。文字列は大文字と小文字を区別しません。
   - 値は、httpデプロイされたアプリケーションのまたは https URL でなければなりません。

   たとえば、 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 出力について詳しくは、『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てください。
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の URL を表示するには

ビルドアクションやアクションなど、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、GitHub 次の操作を行ってアプリの URL を表示してください。

1. InputsStepsワークフロー定義ファイルのアクションのまたはセクションに環境変数を定義します。変数には以下の特性が必要です。

   - にはappurl、またはendpointurl、結合ダッシュ (-)、アンダースコア ()、またはスペース (_) name を含める必要があります。文字列は大文字と小文字を区別しません。
   - 値は、httpデプロイされたアプリケーションのまたは https URL でなければなりません。

   たとえば、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
   ```

   環境変数の定義に関する詳細は、を参照してください変数を定義する。。

2. 変数をエクスポートします。

   たとえば、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[変数をエクスポートして他のアクションでも使用できるようにする。](#)。

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. [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。
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

- ワークフローが自動的に開始されていない場合は、ワークフローの実行を開始します。新しい実行では、ワークフロー図にアプリの URL がクリック可能なリンクとして表示されているはずです。ランの開始について詳しくは、を参照してください[ワークフロー実行の開始](#)。

デプロイメントターゲットの削除

Amazon ECS AWS CloudFormation CodeCatalyst クラスターやスタックなどのデプロイターゲットをコンソールの [環境] ページから削除できます。

> **Important**
>
> デプロイターゲットを削除すると、 CodeCatalyst コンソールからは削除されますが、 AWS そのターゲットをホストするサービスでは引き続き使用できます (まだ存在する場合)。

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以下の場合、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。
- AWS コンソールの Amazon ECS CloudFormation サービスまたは Amazon ECS サービスからスタックまたはクラスターを削除しました。

デプロイターゲットを削除するには

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。環境について詳しくは、を参照してください環境を使用する。
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。

   ターゲットがページから削除されます。

ワークフローでの作業

ワークフローは、継続的インテグレーションと継続的デリバリー (CI/CD) システムの一部としてコードをビルド、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、つまりアクションを定義します。ワークフローでは、ワークフローを開始させるイベント、つまりトリガーも定義されます。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エディターまたは YAML エディター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、ブループリントを使用してプロジェクトを作成してください。各ブループリントには、レビュー、実行、実験が可能な、機能するワークフローが導入されています。

トピック

- ワークフローの作成、編集、削除
- ワークフローステータスの表示
- アクションの使用
- アーティファクトによる作業
- コンピューティング環境とランタイム環境の Docker イメージの操作

- 環境を使用する
- ファイルキャッシュの操作
- パッケージを操作する
- ランの処理
- シークレットの使用
- ソースの操作
- トリガーの使用
- 変数の操作

ワークフローの作成、編集、削除

次の手順を使用して、ワークフローを作成、編集、削除します。

Visual

ビジュアルエディターを使用してワークフローを作成するには

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を配置するソースリポジトリを選択します。`~/.codecatalyst/workflows/`ファイルは選択したリポジトリのフォルダーに保存されます。ソースリポジトリが存在しない場合は、ソースリポジトリを作成します。
6. 「ブランチ」フィールドで、ワークフロー定義ファイルを配置するブランチを選択します。
7. [作成] を選択します。

   Amazon CodeCatalyst はリポジトリとブランチ情報をメモリに保存しますが、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:

a. (オプション) ワークフロー図で、「ソース」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] を選択して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. [+ アクション] (左上) を選択します。アクション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、「アクションの追加」を参照してください。

d. (オプション) ワークフローのプロパティ (右上) を選択します。[ワークフローのプロパティ] ペインが表示されます。ワークフロー名、実行モード、コンピュートを設定します。詳細については、「キュー実行、代替実行、parallel 実行の設定」および「コンピューティング環境とランタイム環境の Docker イメージの操作」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、[ワークフローをコミット] 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、デフォルト名をそのまま使用するか、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そのまま使用するか、独自のメッセージを入力します。

c. 「Repository」と「Branch」では、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、「Create workflow」ダイアログボックスで以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。必要であれば、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はできませんので、慎重に選択してください。

d. [Commit] を選択してワークフロー定義ファイルをコミットします。

YAML

YAML エディターを使用してワークフローを作成するには

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を配置するソースリポジトリを選択します。`~/.codecatalyst/workflows/`ファイルは選択したリポジトリのフォルダーに保存されます。ソースリポジトリが存在しない場合は、ソースリポジトリを作成します。
6. 「ブランチ」フィールドで、ワークフロー定義ファイルを配置するブランチを選択します。
7. [作成] を選択します。

   Amazon CodeCatalyst はリポジトリとブランチ情報をメモリに保存しますが、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:

   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. + アクション (左上) を選択します。アクション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、「アクションの追加」を参照してください。

   d. (オプション) ワークフローのプロパティ (右上) を選択します。[ワークフローのプロパティ] ペインが表示されます。ワークフロー名、実行モード、コンピュートを設定します。詳細については、「キュー実行、代替実行、parallel 実行の設定」および「コンピューティング環境とランタイム環境の Docker イメージの操作」を参照してください。
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、[ワークフローをコミット] 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、デフォルト名をそのまま使用するか、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そのまま使用するか、独自のメッセージを入力します。

    c. 「Repository」と「Branch」では、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、「Create workflow」ダイアログボックスで以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。必要であれば、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はできませんので、慎重に選択してください。

    d. [Commit] を選択してワークフロー定義ファイルをコミットします。

ワークフローを編集するには

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. 編集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. [編集] を選択します。
6. ワークフローを編集します。ビジュアルエディターを使用するには [Visual] を選択し、YAML エディターを使用するには [YAML] を選択します。次のようにできます。

    a. トリガーを編集する。詳細については、「トリガーの使用」を参照してください。

    b. アクションを編集する。詳細については、「アクションの使用」を参照してください。

7. その他のワークフロープロパティを編集する。ワークフローのプロパティ (右上) を選択して、ワークフロー名、実行モード、またはコンピュートを変更します。詳細については、「キュー

実行、代替実行、parallel 実行の設定」および「コンピューティング環境とランタイム環境の Docker イメージの操作」を参照してください。

ワークフローを削除するには

ワークフローを削除すると、ワークフロー定義ファイルと関連する実行が削除されます。AWS削除したワークフローからデプロイされたサービスはすべて手動で削除する必要があります。そうしないと、それらのサービスに対して引き続き料金が発生します。

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. 削除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. [削除] をクリックします。
6. [確認] を選択します。

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をトラブルシューティングしたりできます。 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。実行ステータスはワークフローステータスとは異なります。詳細については、「ワークフロー実行のステータスと詳細の表示」を参照してください。

ワークフローには、次のいずれかのステータスがあります。

- 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が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ない必要があります。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。
- 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、ワークフロー定義に ***n*** 件のエラーメッセージ (または同様のエラーメッセージ) が CodeCatalyst コンソールに表示されます。

  ワークフローを無効としてマークするには、次の条件が true である必要があります。
  - ワークフロー定義ファイルの設定が間違っ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##### ##。
- 非アクティブ – ワークフロー定義は有効ですが、手動で、またはトリガーを使用して自動的に実行することはできません。

  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が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「ワークフローは無効です」というメッセージを修正する方法を教えてください。。

> **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) が指定されている場合、この変更は検出 CodeCatalyst されず、引き続きワークフローを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。

アクションの使用

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論理的な作業単位を定義します。通常、ワークフローには、設定方法に応じて順次またはparallel 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
- アクションの追加、設定、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
- 他のアクションに依存するアクションの設定
- アクションのソースコードを表示する
- アクションバージョンでの作業

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 ラボのアクション
- 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、維持、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 AWS Lambda 関数の呼び出しなどのさまざまな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 公開

  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、「「Amazon S3 パブリッシュ」アクションの追加」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、「「AWS CDK ブートストラップ」アクションの追加」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、「「AWS CDK デプロイ」アクションの追加」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、「AWS Lambda 「呼び出し」アクションの追加」を参照してください。
- AWS CloudFormation スタックのデプロイ

  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、「「 AWS CloudFormation デプロイスタック」アクションの追加」を参照してください。

- Amazon ECS へのデプロイ

  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、「「Amazon ECS にデプロイ」アクションの追加」を参照してください。
- Kubernetes クラスターへのデプロイ

  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、「「Kubernetes クラスターへのデプロイ」アクションの追加」を参照してください。
- Amazon ECS タスク定義をレンダリングする

  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、「「Amazon ECS タスク定義をレンダリング」アクションの追加」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アクションを追加する方法については、「」を参照してくださいアクションの追加。

CodeCatalyst ラボのアクション

CodeCatalyst ラボアクションは、実験的なアプリケーションの証明機関である Amazon CodeCatalyst Labs の一部であるアクションです。 CodeCatalyst ラボアクションは、 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。
- へのデプロイ 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

このアクションでは、 AWS Serverless Application Model () を使用して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、特定のパスのセットの CloudFront キャッシュを無効にします。

- 送信 Webhook アクション

  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 リポジトリに公開します。

- Amazon SNS への発行

  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) リポジトリに公開します。

- 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

  このアクションでは、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 にあります。

ワークフローに CodeCatalyst Labs アクションを追加し、その readme を表示する方法については、「」を参照してくださいアクションの追加。

GitHub アクション

GitHub アクション は、 GitHub ワークフローで使用するために開発された点を除いて、CodeCatalyst アクション によく似ています。 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。

GitHub アクションは、 CodeCatalyst ワークフロー内のネイティブ CodeCatalyst アクションと一緒に使用できます。

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 に記載されている任意の 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 にあります。

詳細については、「 GitHub アクションの追加」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ようにするアクションです。サードパーティーアクションの例は、メンドによって作成されたメンド SCA アクションです。

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 にあります。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。

ワークフローにサードパーティーアクションを追加し、その readme を表示する方法については、「」を参照してくださいアクションの追加。

アクションの追加、設定、削除

アクションをワークフローに追加する方法と削除する方法については、以下のトピックのいずれかを選択してください。

トピック

- アクションの追加
- 「Amazon S3 パブリッシュ」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
- AWS Lambda 「呼び出し」アクションの追加
- 「Amazon ECS タスク定義をレンダリング」アクションの追加
- GitHub アクションの追加

- アクションを削除する

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. 左上部で + 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 Labs、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 AWSがあります。
     - CodeCatalyst Labs アクションには、by CodeCatalyst Labs ラベルがあります。
     - サードパーティーアクションには**####**別ラベルがあり、####はサードパーティーベンダーの名前です。
   - GitHub アクション のキュレートされたリストを表示するには GitHub 、 を選択します。
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。ビジュアルエディタを使用する場合はビジュアル、YAML エディタを使用する場合は YAML を選択します。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 S3 パブリッシュ」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
- 「AWS CDK デプロイ」アクションの追加
- AWS Lambda 「呼び出し」アクションの追加
- 「 AWS CloudFormation デプロイスタック」アクションの追加
- 「Amazon ECS にデプロイ」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
- 「Amazon ECS タスク定義をレンダリング」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。

- GitHub アクションの追加

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。

「Amazon S3 パブリッシュ」アクションの追加

このセクションでは、Amazon S3 パブリッシュアクションをワークフローに追加する方法について説明します。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配置できます。

- 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

トピック

このアクションをいつ使うべきか

このアクションは次の場合に使用します。

- 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。

  たとえば、Amazon S3 でホストしたい静的ウェブサイトを構築するワークフローがあるかもしれません。この場合、ワークフローには、サイトの HTML とサポートファイルをビルドするビルドアクションと、ファイルを Amazon S3 にコピーする Amazon S3 パブリッシュアクションが含まれます。
- Amazon S3 に保存したいファイルを含むソースリポジトリがある。

  たとえば、Amazon S3 に毎晩アーカイブしたい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。

ワークフローの例

以下のワークフロー例には、Amazon S3 パブリッシュ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なドキュメンテーションウェブサイトを構築し、それをホストされている Amazon S3 に公開します。このワークフローは、順番に実行される以下の構成要素で構成されています。

> **Note**
>
> 以下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、ソースリポジトリに変更を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、「トリガーの使用」を参照してください。

- ビルドアクション (BuildDocs) — トリガー時に、アクションは静的なドキュメンテーション Web サイト (mkdocs build) を構築し、関連する HTML MyDocsSite 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。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でのワークフローを使用した構築 CodeCatalyst。
- 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、MyDocsSiteこのアクションはアーティファクト内のサイトをホスティング用の 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 パブリッシュ」アクションの追加

次の手順を使用して、Amazon S3 パブリッシュ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/ 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 [Amazon] を選択します CodeCatalyst。
9. Amazon S3 パブリッシュ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「Amazon S3 公開」アクションリファレンス。このリファレンスには、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) の詳細情報が記載されています。
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。

YAML

YAML エディタを使用して「Amazon S3 パブリッシュ」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 [Amazon] を選択します CodeCatalyst。
9. Amazon S3 パブリッシュ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、「Amazon S3 公開」アクションリファレンス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。

「Amazon S3 パブリッシュ」アクションによって生成される変数

Amazon S3 発行アクションを実行すると、後続のワークフローアクションで使用できる変数が生成されます。詳細については、「Amazon S3 パブリッシュ」アクション変数の「」を参照してください定義済み変数のリスト。

「Amazon S3 パブリッシュ」アクション定義

Amazon S3 パブリッシュアクションは、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、「Amazon S3 公開」アクションリファレンスワークフロー定義リファレンスのを参照してください。

「AWS CDK ブートストラップ」アクションの追加

このセクションでは、AWS CDK ブートストラップアクションをワークフローに追加する方法について説明します。AWS CDK ブートストラップアクションは、 AWS 最新のテンプレートを使用して環境内のブートストラップスタックをプロビジョニングします。ブートストラップスタックがすでに存在する場合、アクションは必要に応じてそれを更新します。ブートストラップスタックがあることは、 AWS アプリをデプロイするための前提条件です。 AWS CDK

ブートストラップについて詳しくは、『開発者ガイド』の「ブートストラップ」を参照してください。AWS Cloud Development Kit (AWS CDK)

トピック

- このアクションをいつ使うべきか
- 仕組み
- 「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
- 前提条件
- 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 bootstrap」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」アクション定義

このアクションをいつ使うべきか

このアクションは、 AWS CDK アプリをデプロイするワークフローがあり、同時にブートストラップスタックをデプロイ (および必要に応じて更新) したい場合に使用します。この場合、AWS CDK アプリケーションをデプロイするワークフローと同じワークフローにブートストラップアクションを追加します。 AWS CDK

以下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。

- すでに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デプロイしていて、その状態を維持したい (更新は行わない)。
- 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いたい。AWS CDK

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 (AWS CDK ツールキットとも呼ばれる) をビルドイメージにダウンロードします。 CodeCatalyst

   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 にバンドルされているため、ダウンロードは行われません。
2. アクションは CDK CLI `cdk bootstrap` を使用してコマンドを実行します。このコマンドは、『開発者ガイド』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 Cloud Development Kit (AWS CDK)

「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る CDK CLI のバージョンを示しています。

> **Note**
>
> デフォルトはオーバーライドできるかもしれません。詳細については、「AWS CDK 「ブートストラップ」アクションリファレンス」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 またはそれ以降	2.99.1

前提条件

AWS CDK ブートストラップアクションを使用する前に、アプリの準備が整っ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は、 AWS CDK ブートストラップの前にアプリを合成します。アプリは、がサポートする任意のプログラミング言語で作成できます。 AWS CDK

AWS CDK アプリケーションファイルが次の場所で入手可能であることを確認してください。

- CodeCatalyst ソースリポジトリ、または
- CodeCatalyst 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。

ワークフローの例

ワークフローの例「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を含むワークフローについては、のを参照してください。

「AWS CDK ブートストラップ」アクションの追加

以下の手順に従って、AWS CDK ブートストラップアクションをワークフローに追加してください。

前提条件

開始する前に、前提条件で説明されているタスクを完了していることを確認してください。

Visual

ビジュアルエディターを使って「AWS CDK bootstrap」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 [Amazon] を選択します CodeCatalyst。
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

   - 「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AWS CDK「ブートストラップ」アクションリファレンス。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. [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。

> **Note**
>
> AWS CDK **`npm install`**ブートストラップアクションがエラーで失敗した場合、「npm install」エラーを修正するにはどうすればいいですか?エラーの修正方法についてはを参照してください。

YAML

YAML エディターを使用して「AWS CDK bootstrap」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 [Amazon] を選択します CodeCatalyst。
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. 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、AWS CDK「ブートストラップ」アクションリファレンス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。

> **Note**
>
> AWS CDK **`npm install`**ブートストラップアクションがエラーで失敗した場合、「npm install」エラーを修正するにはどうすればいいですか?エラーの修正方法についてはを参照してください。

「AWS CDK bootstrap」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を実行すると、後続のワークフローアクションで使用できる変数が生成されます。詳細については、「AWS CDKブートストラップ」アクション変数定義済み変数のリストのを参照してください。

「AWS CDK ブートストラップ」アクション定義

AWS CDK ブートストラップアクションは、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、AWS CDK「ブートストラップ」アクションリファレンスのを参照してください。ワークフロー定義リファレンス

AWS Lambda「呼び出し」アクションの追加

このセクションでは、AWS Lambda呼び出しアクションをワークフローに追加する方法について説明します。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内の各最上位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たくない場合は、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」アクションリファレンス。

トピック

- このアクションを使用するタイミング
- ワークフローの例
- AWS Lambda「呼び出し」アクションの追加
- 「呼び出しAWS Lambda」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、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ビルドを開始する前に、ワークフローで Slack チャンネルBuild startedに通知を送信できます。この場合、ワークフローには Lambda を呼び出して Slack 通知を

送信するAWS Lambda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、デプロイ前にアプリケーションに対して脆弱性スキャンを実行するワークフローが必要になる場合があります。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。

ワークフローの例

> **Note**
>
> 次のワークフロー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提供することを目的としています。

次のワークフローには、デプロイアクションと共に AWS Lambda 呼び出しアクションが含まれます。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を使用してアプリケーションを にデプロイします。ワークフローは、順次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、「トリガーの使用」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、AWS CloudFormationスタックのデプロイ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「 AWS CloudFormation デプロイスタック」アクションの追加。

```
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ます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. 左上部で + 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. 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) コミットする前に、検証を選択してワークフローの 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て、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. 左上部で + 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」アクションリファレンス。
11. (オプション) コミットする前に、検証を選択してワークフローの 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て、もう一度コミットを選択します。

「呼び出しAWS Lambda」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が実行されると、後続のワークフローアクションで使用できる変数が生成されます。詳細については、「」の「AWS Lambdainvoke」アクション変数「」を参照してください定義済み変数のリスト。

AWS Lambda 「呼び出し」アクション定義

AWS Lambda 呼び出しアクションは、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、「」のAWS Lambda 「呼び出し」アクションリファレンス「」を参照してくださいワークフロー定義リファレンス。

「Amazon ECS タスク定義をレンダリング」アクションの追加

このセクションでは、Render Amazon ECS タスク定義アクションをワークフローに追加する方法について説明します。このアクションは、Amazon Elastic Container Service (Amazon ECS) 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提供される 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、`environment`タスク定義のフィールドを環境変数で更新することもできます。

トピック

このアクションをいつ使うべきか

Docker イメージをビルドしてコミット ID やタイムスタンプなどの動的コンテンツを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。

仕組み

Amazon ECS のレンダリングタスク定義アクションは、ワークフローの [ビルド] および [Amazon ECS へのデプロイ] アクションと一緒に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以下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をタグ付けします。たとえば、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} .
          ...
```

   上記のコードでは、docker build -tディレクティブは Docker イメージをビルドし、アクションの実行時にコミット ID でタグ付けすることを指示しています。生成されるイメージ名は以下のようになります。

   MyDockerImage:a37bd7e

2. Render Amazon ECS タスク定義アクションはMyDockerImage:a37bd7e、次のように動的に生成されたイメージ名を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、Render Amazon ECS タスク定義アクションを使用して、次のように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 開発者ガイド』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。

ワークフローの例

以下は、Render Amazon ECS タスク定義アクションとビルドおよびデプロイアクションを含む完全なワークフローの例です。この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、順番に実行される以下のビルディングブロックで構成されています。

- トリガー — このトリガーは、ソースリポジトリに変更を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、「トリガーの使用」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 をタグ付けして、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。でのワークフローを使用した構築 CodeCatalyst
- Render Amazon ECS タスク定義アクション (RenderTaskDef) — ビルドアクションが完了すると、taskdef.jsonこのアクションはソースリポジトリのルートにある既存のものを、正しいコミット ID image を含む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、そのファイルを含む出力アーティファクトを作成します。また、task-definitionレンダリングアクションではという変数が生成され、新しいタスク定義ファイルの名前に設定されます。このアーティファクトと変数は、次のデプロイアクションで使用されます。
- Amazon ECS へのデプロイアクション (DeployToECS) — Amazon ECS のレンダリングタスク定義アクションが完了すると、Amazon ECS へのデプロイアクションは、レンダリングアクション (TaskDefArtifact) によって生成された出力アーティファクトを探し、task-definition-*random-string*.jsonその中の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 task-definition-*random-string*.json サービスはファイル内の指示に従って Amazon ECS タスクおよび関連する Docker イメージコンテナを Amazon ECS クラスター内で実行します。

```
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 タスク定義をレンダリング」アクションの追加

次の手順を使用して、Render Amazon ECS タスク定義アクションをワークフローに追加します。

前提条件

始める前に、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 タスク定義をレンダリング」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 [Amazon] を選択します CodeCatalyst。
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 (+) を選択してアクションをワークフローダイアグラムに追加し、設定ペインを開きます。

     または

   - [Amazon ECS タスク定義をレンダリング] 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「Amazon ECS タスク定義をレンダリングする」アクションリファレンス。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. [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 エディタを使用して「Amazon ECS タスク定義をレンダリング」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 [Amazon] を選択します CodeCatalyst。
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 (+) を選択してアクションをワークフローダイアグラムに追加し、設定ペインを開きます。

     または

   - [Amazon ECS タスク定義をレンダリング] 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、「Amazon ECS タスク定義をレンダリングする」アクションリファレンス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。

次のステップ

レンダリングアクションを追加したら、の手順に従って Deploy to Amazon ECS アクションをワークフローに追加します。「Amazon ECS にデプロイ」アクションの追加デプロイアクションを追加する際、次の操作を行います。

1. デプロイアクションの「インプット」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。これには、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、「アーティファクトによる作業」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* はレンダリングアクションの名前 (例:) です。RenderTaskDefレンダーアクションは、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変数の操作。

   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例を参照してください。

更新したタスク定義ファイルを表示する

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。

Render Amazon ECS タスク定義アクションが処理した後に、更新されたタスク定義ファイルの名前を表示するには。

1. 完了したレンダリングアクションを含む実行を検索します。
   a. https://codecatalyst.aws/ CodeCatalyst で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. ワークフローダイアグラムで、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。と似ていますtask-definition--259-0a2r7gxlTF5X-.json。

更新したタスク定義ファイルの内容を表示するには

1. 完了したレンダリングアクションを含む実行を検索します。
   a. https://codecatalyst.aws/ CodeCatalyst で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. ワークフローランの上部にある [ビジュアル] と [YAML] の横にある [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、「制作者」列にレンダリングアクションの名前が設定されます。
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 タスク定義をレンダリング」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を実行すると、後続のワークフローアクションで使用できる変数が生成されます。詳細については、「Amazon ECS タスク定義をレンダリング」アクション変数の 定義済み変数のリスト「」を参照してください。

「Amazon ECS タスク定義をレンダリング」アクション定義

Render Amazon ECS タスク定義アクションは、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、「Amazon ECS タスク定義をレンダリングする」アクションリファレンスのを参照してください。ワークフロー定義リファレンス

GitHub アクションの追加

GitHub アクションは、 GitHub ワークフローで使用するために開発されたという点を除けば、CodeCatalyst アクションによく似ています。 GitHub アクションについて詳しくは、GitHub アクションのドキュメントを参照してください。

GitHub CodeCatalyst CodeCatalyst アクションはワークフロー内のネイティブアクションと一緒に使用できます。

GitHub CodeCatalyst アクションをワークフローに追加するには 2 つの方法があります。

- GitHub CodeCatalyst コンソールの厳選されたリストからアクションを選択できます。 GitHub 一般的なアクションがいくつか用意されています。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。
- GitHub CodeCatalyst 使用したいアクションがコンソールにない場合は、GitHub アクションアクションを使用して追加できます。

  GitHub 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。

  以下は、GitHub GitHubスーパーリンターアクションをラップするアクションアクションの例です。

```
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`) が Super-Linter アクション (で識別`github/super-linter@v4`) 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

  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。

GitHub キュレーションされたアクションもそうでないアクションも、前の例で示したように、GitHub すべてのアクションは Actions action () `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。

トピック

- GitHub アクションはアクションとどう違うのですか CodeCatalyst？
- GitHub CodeCatalyst アクションはワークフロー内の他のアクションと相互作用できますか?
- GitHub どのアクションを使用できますか?
- GitHub でのアクションの制限事項 CodeCatalyst
- GitHub アクション (大まかなステップ) を追加する方法を教えてください。
- GitHub GitHubアクションは実行されますか?
- GitHub ワークフローも使用できますか?
- チュートリアル:アクションを使った lint コード GitHub
- 「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
- GitHub アクション出力パラメータの操作

GitHub アクションはアクションとどう違うのですか CodeCatalyst？

GitHub CodeCatalyst ワークフロー内で使用されるアクションには、 CodeCatalyst CodeCatalyst アクションと同じレベルのアクセスや機能 (環境や課題など) AWS との統合はありません。

GitHub CodeCatalyst アクションはワークフロー内の他のアクションと相互作用できますか?

はい。たとえば、 GitHub CodeCatalyst アクションは他のアクションによって生成された変数を入力として使用でき、 CodeCatalyst 出力パラメータやアーティファクトをアクションと共有することもできます。詳細については、「 GitHub アクション出力パラメータの操作」を参照してください。

GitHub どのアクションを使用できますか?

GitHub CodeCatalyst コンソールから利用できるすべてのアクションと、GitHubMarketplace GitHub で利用できるすべてのアクションを使用できます。Marketplace GitHub アクションを使用する場合は、以下の制限に留意してください。

GitHub でのアクションの制限事項 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 イメージの操作

- GitHub githubの内部的にコンテキストに依存するアクションや、 GitHub固有のリソースを参照するアクションは動作しません。 CodeCatalystたとえば、以下のアクションは動作しません。 CodeCatalyst
  - GitHub リソースの追加、変更、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するアクションや、で課題を作成するアクションなどがあります GitHub。
  - https://github.com/actions にリストされているほとんどすべてのアクションです。
- 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 (root) が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。(この記事を書いている時点では、ユーザー 1001 は作業していますが GitHub、作業はしていません)。 CodeCatalyst詳細については、Dockerfile のアクションサポートの USER トピックを参照してください。 GitHub

GitHub CodeCatalyst コンソールで使用できる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 (大まかなステップ) を追加する方法を教えてください。

GitHub CodeCatalyst アクションをワークフローに追加する手順の概要は次のとおりです。

1. CodeCatalyst プロジェクトでは、ワークフローを作成します。ワークフローは、アプリケーションのビルド、テスト、デプロイの方法を定義する場所です。詳細については、「でのワークフロー入門 CodeCatalyst」を参照してください。
2. ワークフローでは、 GitHub キュレーションされた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。
   - キュレーションされたアクションを追加することを選択した場合は、そのアクションを設定します。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。
   - キュレーションされていないアクションを追加する場合は、GitHubアクションアクション内にアクションの YAML コードを貼り付けます GitHub 。このコードは、GitHubMarketplace GitHub で選択したアクションの詳細ページにあります。このコードを動作させるには、コード

を少し修正する必要があるでしょう 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) ワークフロー内に、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、「のワークフローによるビルド、テスト、デプロイ CodeCatalyst」を参照してください。
5. ワークフローは手動で、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 GitHubアクションは実行されますか?

いいえ。 GitHub アクションは CodeCatalyst、 CodeCatalystのビルドマシンを使用して実行されます。

GitHub ワークフローも使用できますか?

いいえ。

チュートリアル:アクションを使った lint コード GitHub

このチュートリアルでは、 GitHub スーパーリンターアクションを Amazon ワークフローに追加します。 CodeCatalyst 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、疑わしい構成がある箇所を見つけて、結果をコンソールに出力します)。 CodeCatalystリンターをワークフローに追加したら、ワークフローを実行してサンプル Node.js アプリケーション () をリントします。`app.js`次に、報告された問題を修正し、ワークフローを再実行して、修正がうまくいったかどうかを確認します。

> **Tip**
>
> Super-Linter を使用してテンプレートなどの YAML ファイルをリントすることを検討してください。AWS CloudFormation

トピック

前提条件

始める前に、次のものが必要です。

- CodeCatalyst が接続されたスペースAWS アカウント。詳細については、「AWS Builder ID ユーザーをサポートするスペースの作成」を参照してください。
- という名前の、 CodeCatalyst スペース内の空のプロジェクト`codecatalyst-linter-project`。「最初から開始」オプションを選択して、このプロジェクトを作成します。

  詳細については、「Amazon で空のプロジェクトを作成する 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/ 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`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] に、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、app.jsソースリポジトリにファイルを追加します。には、app.jsリンターが見つけるような誤りがいくつかある関数コードが含まれています。

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、codecatalyst-linter-source-repositoryリポジトリを選択します。
4. [ファイル] で [ファイルを作成] 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。他のオプションはデフォルトのままにします。
7. [Commit] (コミット) を選択します。

   これで、という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したときに 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以下のビルディングブロックで構成されています。

- トリガー — このトリガーは、ソースリポジトリに変更を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、「トリガーの使用」を参照してください。
- 「GitHub アクション」アクション — トリガーされると、GitHub Actions アクションは Super-Linter アクションを実行し、次にソースリポジトリ内のすべてのファイルを検査します。リンターが問題を発見すると、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] には、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
```

上記のコードでは、この手順の次のステップで説明されているように、Super-Linter *github-action-code*アクションコードに置き換えます。

9. Marketplace スのスーパーリンターページにアクセスしてください。 GitHub
10. `steps:`(小文字) でコードを探し、(大文字) CodeCatalyst の下のワークフローに貼り付けます。`Steps:`

    次のコードに示すように、 GitHub CodeCatalyst 標準に準拠するようにアクションコードを調整します。

    これで、 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) コミットする前に [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。
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て、もう一度 [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。ワークフローの上部に定義されているトリガーにより、ワークフローの実行が自動的に開始されます。

進行中のワークフローを表示するには

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. 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まで待ってください。リンターがコード内の問題を発見したため、この失敗は予想されます。
5. CodeCatalyst コンソールは開いたままにして、に進んでください。ステップ 4: スーパーリンターが発見した問題の修正

ステップ 4: スーパーリンターが発見した問題の修正

Super-Linterは、README.mdソースリポジトリに含まれるファイルだけでなく、app.jsコードにも問題を発見しているはずです。

リンターが発見した問題を解決するには、

1. CodeCatalyst コンソールで [ログ] タブを選択し、次に [Lint コードベース] を選択します。

   Super-Linter アクションによって生成されたログが表示されます。
2. Super-Linterのログで、問題の発生箇所がわかる 90 行目までスクロールします。ログは以下のようになります。

```
/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. app.jsREADME.mdソースリポジトリでとを修正し、変更をコミットしてください。

> **Tip**
>
> を修正するにはREADME.md、次のようにコードブロックに追加しますmarkdown。
>
> ```
> ```markdown
> Setup examples:
> ...
> ```
> ```

変更を加えると、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までお待ちください。すべての問題を解決すれば、ワークフローは成功するはずです。

クリーンアップ

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

でクリーンアップするには CodeCatalyst

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. 削除codecatalyst-linter-source-repository。
3. 削除codecatalyst-linter-workflow。

このチュートリアルでは、Super-Linter GitHub CodeCatalyst アクションを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、「GitHub アクションの追加」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:アクションを使った lint コード GitHub。

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. [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」アクションリファレンス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」アクションリファレンスワークフロー定義リファレンス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 GitHub アクションの追加。

> **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」アクションリファレンス。このリファレンスでは、YAML GitHub工ディタとビジュアル工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」アクションリファレンス

    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。

GitHub アクション出力パラメータの操作

GitHub CodeCatalyst 出力パラメータはワークフローに統合できます。

> **Note**
>
> 出力パラメータの別名は「変数」です。 GitHub ドキュメントでは「出力パラメーター」という用語が使われているので、この用語も使用します。

トピック

- GitHub 出力パラメータをエクスポートして他のアクションでも使用できるようにする。
- GitHub出力パラメータを参照する。

GitHub 出力パラメータをエクスポートして他のアクションでも使用できるようにする。

GitHub 次の手順に従って出力パラメータをエクスポートし、 CodeCatalyst 他の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。手順については、「ワークフローを編集するには」を参照してください。
2. GitHub エクスポートする出力パラメータを生成するアクションアクション に、OutputsVariables次のような基礎となるプロパティを含む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:

- *id: GitHub steps############################ ID*。
- *##############*。 GitHub

例

次の例は、 GitHub という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出力パラメータを参照する。

GitHub 出力パラメータを参照するには、次の手順に従います。

GitHub 出力パラメータを参照するには:

1. GitHub 出力パラメータをエクスポートして他のアクションでも使用できるようにする。 のステップを完了します。

   これで、 GitHub 出力パラメータを他のアクションで使用できるようになりました。
2. Variables出力パラメータの値を書き留めておきます。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、 CodeCatalystGitHub 出力パラメータを生成するアクションの名前 (GitHub nameidアクションのまたはは使用しないでください)。
   - *output-name* に、先ほど書き留めておいた出力パラメータの値が入っていますVariables。

例

```
BuildActionB:
  Identifier: aws/build@v1
  Configuration:
    Steps:
      - Run: echo ${MyGitHubAction.random-color-generator_SELECTEDCOLOR}
```

コンテキスト付きの例

次の例はGitHubActionA、SELECTEDCOLOR変数を設定して出力し、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}
```

アクションを削除する

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ーを使用してアクシ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. 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、縦向きの省略記号アイコンを選択し、[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. [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アクシ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. 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 のセクションを探します。

   セクションを選択し、キーボードの Delete 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. [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。

カスタムアクションの開発

アクション開発キット (ADK) を使用して、 CodeCatalyst ワークフローで使用するカスタムアクションを開発できます。その後、 CodeCatalyst アクションをアクションカタログに公開して、CodeCatalyst 他のユーザーが自分のワークフローで表示して使用できるようにします。

アクション (高レベルタスク) を開発、テスト、公開するには

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. CodeCatalyst アクションコードを保存する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特定されます。
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のビルド、テスト、リリースに必要なツールとライブラリを入手してください。
5. ローカルコンピュータでアクションをビルドし、 CodeCatalyst変更をリポジトリにプッシュします。
6. ローカルでユニットテストを行ってアクションをテストし、ADK が生成したワークフローをで実行します。 CodeCatalyst
7. コンソールの [公開] ボタンを選択して、 CodeCatalyst アクションをアクションカタログに公開します。 CodeCatalyst

詳細な手順については、Amazon CodeCatalyst アクション開発キット開発者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

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、異なるアクショングループ間の依存関係を設定することもできます。

> **Note**
>
> アクショングループを他のアクションやアクショングループ内にネストすることはできません。

次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 を選択すると YAML のインストラクションが表示されます。

YAML

アクショングループを定義するには:

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. に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/ecs-deploy@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、action-group-nameアクションのプロパティの説明を参照してくださいワークフロー定義リファレンス。

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. [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。

例:2 つのアクショングループの定義

次の例は、BuildAndTestと 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アクショングループには 2 つのアクション (BuildとTest) が含まれ、Deployアクショングループには 2 つのアクション (DeployCloudFormationStackとDeployToECS) も含まれています。

```
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アクションの設定

デフォルトでは、アクションをワークフローに追加すると、ビジュアルエディターに並べて追加されます。つまり、ワークフローの実行を開始すると、アクションがparallel 実行されます。アクションを順番に実行させたい (そしてビジュアルエディターでは縦に表示したい) 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。たとえば、Testアクションに依存するようにアクションを設定して、Buildビルドアクションの後にテストアクションが実行されるようにすることができます。

アクション間、アクショングループ間、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。また、1 one-to-many つのアクションが他の複数のアクションに依存して起動するように依存関係を設定することもできます。依存関係の設定に関するガイドラインを参照して、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てください。

トピック

依存関係を設定する

次の手順に従って、ワークフロー内のアクション間の依存関係を設定します。

Visual

ビジュアルエディターを使用して依存関係を設定するには

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. 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またはアクショングループを指定します。

   「依存する」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アクションの設定
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. [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依存関係を設定するには

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. 別のアクションに依存するアクションには、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、「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の設定に関するガイドライン。詳細については、ワークフロー定義リファレンス for your action DependsOn の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. [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。

依存関係の設定に関するガイドライン

依存関係を設定するときは、以下のガイドラインに従ってください。

- アクションがアクショングループ内にある場合、そのアクションは同じアクション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 階層の同じレベルにある他のアクションやアクショングループに依存できますが、別のレベルでは依存できません。

例

以下の例は、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

以下の例は、TestBuildDependsOnプロパティを使用するアクションに依存するように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アクションに依存するようにアクショングループを設定する方法を示しています。FirstActionアクションとアクショングループは同じレベルに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:あるアクショングループが別のアクショングループに依存するように設定する

次の例は、DeployGroupBuildAndTestGroupアクショングループをそのアクショングループに依存するように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に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:アクショングループが複数のアクションに依存するように設定する

次の例は、アクション、DeployGroupアクション、FirstActionSecondActionBuildAndTestGroupおよび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方法を示しています。DeployGroupこれは、、FirstActionSecondAction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:
      ...
```

アクションのソースコードを表示する

アクションのソースコードを表示して、危険なコード、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。

次の手順に従って、CodeCatalyst、CodeCatalyst ラボ、またはサードパーティの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 スのアクションのページにアクセスしてください。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確認できます。

> **Note**
>
> ビルド、テスト、 CodeCatalyst GitHub アクションのソースコードは表示できません。

> **Note**
>
> AWS GitHub アクションまたはサードパーティアクションのアクションコードをサポートまたは保証するものではありません。

アクションのソースコードを表示するには

1. https://codecatalyst.aws/ 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. コードを表示したいアクションを探します。
   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. 任意のワークフローの名前を選択するか、ワークフローを作成します。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、編集、削除。
   c. [編集] を選択します。
   d. 左上の [+ 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、[CodeCatalyst Labs] CodeCatalyst、[サードパーティのアクション] 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 (+) は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にある [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 S3 とは何ですか?」を参照してください。Amazon シンプルストレージサービスユーザーガイドにあります。
5. コードを調べて、品質とセキュリティに対する期待に応えていることを確認してください。

アクションバージョンでの作業

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、 CodeCatalyst Amazonは次の形式で完全版をワークフロー定義ファイルに追加します。

v*major*.*minor*.*patch*

例:

```
My-Build-Action:
  Identifier: aws/build@v1.0.0
```

Identifierプロパティのフルバージョンを短くして、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。

たとえば、次のように指定したとします。

```
My-CloudFormation-Action:
  Identifier: aws/cfn-deploy@v1.0
```

... 最新のパッチバージョンがの場合1.0.4、アクションはを使用します1.0.4。たとえば、新しいバージョンがリリースされた場合1.0.5、アクションはを使用します1.0.5。たとえば、マイナーバージョンがリリースされた場合でも1.1.0、アクションは引き続き使用されます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。

トピック

- 使用するアクションバージョンの指定
- アクションのどのバージョンが使用できるかを判断する

使用するアクションバージョンの指定

次の手順に従って、ワークフローで使用したいアクションのバージョンを指定してください。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。

アクションには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 を選択すると YAML のインストラクションが表示されます。

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/ 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. バージョンを編集したいアクションを探します。
8. `Identifier`アクションのプロパティを探し、バージョンを次のいずれかに設定します。

   - アクション識別子 @v *`major`* — この構文を使用してワークフローに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します。
   - *`######## @v ##### minor`* — この構文を使用すると、ワークフローで特定のマイナーバージョンが使用され、最新のパッチバージョンが自動的に選択されます。
   - *`######## @v ##### ##### patch`* — この構文を使用して、ワークフローに特定のパッチバージョンを使用するようにします。

   > **Note**
   >
   > どのバージョンがあるかわからない場合は、を参照してくださいアクションのどのバージョンが使用できるかを判断する。

   > **Note**
   >
   > メジャーバージョンは省略できません。

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. [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。

アクションのどのバージョンが使用できるかを判断する

次の手順に従って、ワークフローで使用できるアクションのバージョンを判断してください。

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/ 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. 表示したいバージョンのアクションを探します。

   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. 任意のワークフローの名前を選択するか、ワークフローを作成します。ワークフローの作成方法については、を参照してくださいワークフローの作成、編集、削除。

   c. [編集] を選択します。

   d. 左上の [+ 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 [Labs] CodeCatalyst、[CodeCatalystLabs]、およびサードパーティのアクションを表示するか、[GitHub厳選されたアクション] を選択しますGitHub。

   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 (+) は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にある「バージョン」ドロップダウンリストを選択すると、アクションの使用可能なバージョンのリストが表示されます。

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ーの説明が表示されます。

アーティファクトによる作業

アーティファクトはワークフローアクションの出力で、通常はフォルダーまたはファイルのアーカイブで構成されます。アーティファクトはアクション間でファイルや情報を共有できるので重要です。

たとえば、*sam-template.yml*ファイルを生成するビルドアクションがあって、デプロイアクションでそれを使用したい場合があります。このシナリオでは、アーティファクトを使用して、`sam-template.yml`ビルドアクションがデプロイアクションと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) sam-template.yml はファイルを生成し、MYARTIFACTそれをという出力アーティファクトに追加します。後続のデプロイアクション (DeployAction) MYARTIFACT は入力としてを指定し、sam-template.ymlファイルへのアクセスを許可します。

トピック

- 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?
- ワークフロー間でアーティファクトを共有できますか?
- 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
- 例

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 Outputs Inputs コードのおよびセクションでアーティファクトを指定しなくても、アクション間でアーティファクトを共有できます。そのためには、コンピュート共有を有効にする必要があります。コンピュートシェアリングと、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詳しくは、を参照してくださいアクション間での計算の共有。

> **Note**
>
> `Outputs`と`Inputs`セクションが不要になるため、ワークフローの YAML コードを簡略化できますが、この機能には制限があり、有効にする前に知っておく必要があります。これらの制限については、を参照してください[コンピュートシェアリングに関する考慮事項](#)。

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。ただし、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。

出力アーティファクトの定義

次の手順に従って、アクションに出力させたい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も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。使用しているアクションがアーティファクト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ーの説明を実行し、アクションに「出力」タブの「出力アーティファクト」ボタンが含まれているかどうかを確認してください。「はい」の場合、出力アーティファクトはサポートされています。

Visual

ビジュアルエディター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/ 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. 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. [アーティファクトを追加] を選択し、以下のように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るアーティファクトの名前を指定します。Artifact 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にスペース、ハイフン、その他の特殊文字を使用することはできません。

    例を含む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。アーティファクトによる作業

    build によって生成されるファイル

    CodeCatalyst アクションによって出力されるアーティファクトに含まれるファイルを指定します。これらのファイルは、ワークフローアクションの実行時に生成され、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配置でき、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。グロブ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

例を含む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による作業。

> **Note**
>
> 検索するアーティファクトまたはソースを示すプレフィックスをファイルパスに追加する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. [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出力アーティファクトを定義するには

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. ワークフローアクションに、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、「例」を参照してください。詳細については、該当するアクションの ワークフロー定義リファレンス「」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. [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。

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、そ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必要があります。アクションによっては、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。詳細については、アクションのを参照してください。ワークフロー定義リファレンス

> **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。

次の手順に従って、別のアクション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します。

前提条件

始める前に、他のアクションからアーティファクトを出力した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、他のアクションでも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/ 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. 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したい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すでに定義されている必要があります。

   入力アーティファクトを何も指定しない場合は、下に少なくとも 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`

   例を含む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。アーティファクトによる作業

   > **Note**
   >
   > 「アーティファクト-オプション」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ー)、または YAML を検証したときにエラーが発生する場合 (YAML エディター) は、アクションが 1 つの入力しかサポートしていないことが原因である可能性があります。この場合は、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. [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

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 エディター)

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. アーティファクトを入力として指定するアクションに、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、「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. [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。

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あるファイルがあり、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、次の手順を実行してください。

> **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に、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 *artifact-name build-output ########################*。file.txt

その他の例については、「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、以下のガイダンスを参照してください。

変数の参照方法に関するガイダンス:

- アクションに項目が 1 つしか含まれていない場合 Inputs (たとえば、入力アーティファクトが 1 つだけ含まれ、ソースがない場合など)、プレフィックスを省略して、アーティファクトのルートを基準としたファイルパスだけを指定できます。
- 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。主入力はかWorkflowSource、リスト内の最初の入力アーティファクト (存在しない場合) 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のプレフィックス	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`

その他すべてのアクション	`$CATALYST_SOURCE_DIR_ artifact-name /`
	
	または
	
	`/artifacts/ current-action-name /artifact-name /(##### $CATALYST_SOURCE_DIR_ ######## ########## )`
		`$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次の 2 種類があります。

- ソースアーティファクト — 実行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 — Outputs ワークフローの設定ファイルの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/ 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. ワークフロー名の下にある [Runs] 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. 実行の名前の下にある [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 [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。ファイル名はランダムな 7 文字で構成されています。
9. 任意の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。

例

以下の例は、ワークフロー定義ファイル内のアーティファクトを出力して参照する方法を示しています。

トピック

例:アーティファクトを出力する

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、呼び出されたアーティファクトを出力しBuildActionA、ARTIFACT4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、ART5 and という名前の 2 つのアーティファクトを出力しBuildActionC、(ART6in artifact) と (artifact では) in file5.txt (underART5) 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。file6.txt ART6 BuildActionD 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**
>
> $CATALYST_SOURCE_DIR_ART5この例ではプレフィックスが使用されていますが、省略してもかまいません。これは主入力だからです*ART5*。主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:1 つのアーティファクト内のファイルを参照する

次の例は、ART7 in という名前のアーティファクトを 1 つ出力しBuildActionE、次に (under) の file7.txt (アーティファクトで ART7BuildActionF) 参照する方法を示しています。Steps

参照では、build-outputでの場合のようにディレクトリの前に $CATALYST_SOURCE_DIR_*artifact-name* ##########################。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、で指定されている項目が 1 つしかないためです。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:アーティファクト内のファイルがある場合の参照 WorkflowSource

次の例は、ART8 in という名前のアーティファクトを 1 つ出力しBuildActionG、次に (under) の file8.txt (アーティファクトで ART8BuildActionH) 参照する方法を示しています。Steps

で説明したように、参照には $CATALYST_SOURCE_DIR_ *artifact-name* プレフィックスが必要である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、複数の項目 Inputs (ソースとアーティファクト) が指定されているため、ファイルを探す場所を示すプレフィックスが必要だからです。

> **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
```

コンピューティング環境とランタイム環境の Docker イメージの操作

CodeCatalyst ワークフローでは、 CodeCatalyst 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およびランタイム環境のイメージを指定できます。

コンピュートとは、 CodeCatalyst ワークフローアクションを実行するために管理および管理され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。

ランタイム環境イメージは、 CodeCatalystその中でワークフローアクションを実行する 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、ワークフローアクションで必要となる可能性がある追加ツール (Node.js AWS CLI、.tar など) が含まれます。

トピック

コンピュートでの作業

コンピュートとは、 CodeCatalyst ワークフローアクションを実行するために管理・保守され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**
>
> compute 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どのアクションのプロパティとしても定義することはできません。同様に、compute をアクションのプロパティとして定義しても、ワークフローでは定義できません。

トピック

コンピュートタイプについて

CodeCatalyst には以下のコンピュートタイプがあります。

- 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 はアクションの開始速度を最適化します。Lambda では、起動時の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が速くなります。Lambda では、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もあります。影響がある場合は、Amazon EC2 コンピューティングタイプを使用してください。

- Lambda は、指定されたレジストリ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 root 権限を必要とするツールをサポートしていません。yumやなどのツールにはrpm、Amazon EC2 コンピューティングタイプまたはルート権限を必要としないその他のツールを使用してください。
- 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次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 の発行、 AWS CDK ブートストラップ、 AWS CDK デプロイ、AWS Lambda 呼び出し、アクション。 GitHub GitHub アクションアクション内で実行されている Docker CodeCatalyst GitHub ベースのアクションも Lambda compute では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 は外部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際、インストールまたは書き込み先となるようにツールを再設定できます。/tmpインストールするビルドアクションがある場合はnpm、インストール先とな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。

コンピュートフリートについて

CodeCatalyst 以下のコンピュートフリートを提供しています。

- 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。支払いが発生するのは、アクションを実行した分数だけです。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 また、Amazon EC2 を搭載したマシンを含むプロビジョニング済みフリートも提供しています。これらのマシンは Amazon EC2 によって管理されています。 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専用マシンのセットを設定します。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る状態になっています。プロビジョニングされたフリートでは、マシンは常に稼働しており、プロビジョニングされている間はコストが発生します。

フリートを作成、更新、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 以下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

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.Arm64.2XLarge`	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

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プロパティ

プロビジョニングされたフリートには以下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。現在 Windows をサポートしていないアクションもあります。

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
			Windows Server 2022

容量

フリートに割り当てられるマシンの初期数。これにより、parallel 実行できるアクションの数が定義されます。

スケーリングモード

アクションの数がフリートの容量を超えたとき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

必要に応じて追加のマシンをセットアップし、新しいアクションの実行に応じて自動的にスケールアップし、アクションが終了すると基本容量までスケールダウンします。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、編集、削除

以下の手順に従って、プロビジョニングされたフリートを作成、編集、削除します。

> **Note**
>
> プロビジョニングされたフリートは、2 週間使用されない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有効化されますが、再有効化すると遅延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、次に [Compute] を選択します。
2. [プロビジョニング済みフリートの作成] を選択します。
3. 「プロビジョニング済みフリート名」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] 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. キャパシティ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ト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、次に [Compute] 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、編集するフリートを選択します。
3. [編集] を選択します。
4. キャパシティ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 `Fleet` コードからプロパティを削除して、すべてのアクションから削除してください。削除後もプロビジョニングされたフリートを参照し続けるアクションは、次にアクションを実行したときに失敗します。

1. ナビゲーションペインで [CI/CD] を選択し、次に [Compute] 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] をクリックします。
4. **`delete`**と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ト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。プロビジョニングされたフリートを代わりに使用

したり、別のオンデマンドフリートなど、使用したりするにはLinux.x86-64.2XLarge、以下の手順に従ってください。

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、編集、削除」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別の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/ 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. 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. 「コンピュートフリート」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専用マシンのセットを設定します。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る状態になっています。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プロパティ

を省略した場合、Fleetデフォルトはです。Linux.x86-64.Large

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. [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

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、編集、削除」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別の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/ 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 を選択してください。
7. プロビジョニングしたフリートまたは新しいフリートタイプを割り当てたいアクションを探します。
8. Computeアクションでプロパティを追加し、フリートまたはオンデマンドフリートタイプの名前を設定しますFleet。詳細については、Fleetビルドとテストアクションのリファレンスアクションの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. [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。

アクション間での計算の共有

トピック

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別々のインスタンスで実行されます。この動作により、アクションが分離され、入力の状態が予測可能になります。デフォルトの動作では、アクション間でファイルや変数などのコンテキストを共有するための明示的な設定が必要です。

コンピュートシェアリングは、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上で実行できる機能です。コンピュートシェアリングを使用すると、インスタンスのプロビジョニングにかかる時間が短くなるため、ワークフローの実行時間を短縮できます。ワークフローを追加設定しなくても、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。

コンピュートシェアリング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が実行される間予約されます。ワークフローの実行が完了すると、インスタンスの予約は解除されます。

共有コンピュート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 `Compute` 内の属性を使用して、アクションのフリートとコンピュート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ーを使用してコンピュート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、コンピュートタイプを EC2 に設定し、コンピュートシェアリングを有効にします。

> **Note**
>
> コンピュートシェアリングは、コンピュートタイプが EC2 に設定されている場合のみサポートされ、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。コンピュートフリート、コンピュートタイプ、およびプロパティの詳細については、を参照してくださいコンピュートでの作業。

> **Note**
>
> 無料利用枠を利用していて、ワークフロー定義 YAML `Linux.x86-64.XLarge` `Linux.x86-64.2XLarge` でまたはフリートを手動で指定しても、アクションはデフォルトのフリート () `Linux.x86-64.Large` で引き続き実行されます。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。

コンピュートシェアリングをオンにすると、ワークフローソースを含むフォルダーが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て、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、コンピュートシェアリング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接続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ーを共有したい場合は、ファイルキャッシュを検討してください。詳細については、「アーティファクトによる作業」および「ファイルキャッシュの操作」を参照してください。

ワークフロー定義ファイルが置かれているソースリポジトリはラベルで識別されます。`WorkflowSource`コンピュートシェアリングを使用している間、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、ワークフロー実行中の後続のアクションで自動的に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、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ーに加えられた変更は、ワークフロー内の後続のアクションでも表示されます。コンピュートシェアリングを使用しない場合と同様に、どのワークフローアクションでもワークフローソースフォルダー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**
>
> コンピュート共有ワークフローでは、アクションの順序を厳密に指定する必要があるため、parallel アクションは設定できません。出力アーティファクトはシーケンス内のどのアクションでも設定できますが、入力アーティファクトはサポートされていません。

コンピュートシェアリングに関する考慮事項

コンピュートシェアリングを使用してワークフローを実行することで、ワークフローの実行時間を短縮し、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。以下の点を考慮して、コンピュートシェアリングの使用が自分のシナリオに適しているかどうかを判断してください。

	コンピュートシェアリング	コンピュート共有なし
コンピューティングタイプ	Amazon EC2	Amazon EC2、AWS Lambda
インスタンスプロビジョニング	アクションは同じインスタンスで実行されます。	アクションは別々のインスタンスで実行されます。
オペレーティングシステム	Amazon Linux 2	Amazon リナックス 2、ウィンドウズサーバー 2022 (ビルドアクションのみ)
ファイルの参照	`$CATALYST_SOURCE_DIR_WorkflowSource` 、`/sources/WorkflowSource/`	`$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parallel 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 () `WorkflowSource` へのアクセス	共有アーティファクトの出力にアクセスする (追加の設定が必要)

コンピュートシェアリングを有効にする

次の手順に従って、ワークフローのコンピュートシェアリングを有効にします。

Visual

ビジュアルエディターを使用してコンピュートシェアリングを有効にするには

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。
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. [コンピュートタイプ] 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) コンピュートフリート-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したい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ことも、プロビジョニングされたフリートを作成して選択することもできます。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、編集、削除」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ト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ト共有を有効にすると、ワークフロー内のアクションを同じフリートで実行できます。
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、「キュー実行、代替実行、parallel 実行の設定」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て、もう一度 [コミット] を選択します。

YAML

YAML エディターを使用してコンピュートシェアリングを有効にするには

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。
5. [編集] を選択します。
6. YAML を選択します。
7. コンピュートシェアリングをオンにして、SharedInstanceフィールドを「」TRUE と「」に設定しますType。EC2ワークフローアクションの実行に使用したいコンピュート群に設定しますFleet。オンデマンドフリートを選択することも、プロビジョニングされたフリートを作成して選択することもできます。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、編集、削除」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ト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 に、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. [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。

例

トピック

- 例:Amazon S3 パブリッシュ

例:Amazon S3 パブリッシュ

以下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。最初に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シェアリングを使用する方法です。コンピュートシェアリングでは、キャッシュされたものにアクセスできるため、入力アーティファクトは不要です。`WorkflowSource`さらに、Build アクションの出力アーティファクトは不要になります。S3 Publish アクションは、`DependsOn`明示的な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。

- コンピュートシェアリングがなければ、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
```

- SharedInstanceコンピュートシェアリングをに設定して使用するとTRUE、同じインスタンスで複数のアクションを実行し、1 つのワークフローソースを指定することで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、指定することもできません。

```
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 イメージの操作

ランタイム環境イメージは、 CodeCatalystその中でワークフローアクションを実行する 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、ワークフローアクションで必要となる可能性がある追加ツール (Node.js AWS CLI、.tar など) が含まれます。

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管理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

アクティブイメージ

アクティブイメージは、 CodeCatalyst プリインストールされたツールによって完全にサポートされているランタイム環境イメージです。現在、アクティブイメージは 2024 年 3 月にリリースされたものと 2022 年 11 月にリリースされたものの 2 セットです。

アクションが 2024 年 3 月と 2022 年 11 月のどちらの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たビルドアクションとテストアクションには、2024 年 3 `Container` 月のイメージを明示的に指定するセクションが YAML 定義に含まれます。`Container`オプションで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アクションとテストアクションは YAML *`Container`*定義にセクションを含まないため、2022 年 11 月のイメージが使用されます。2022 年 11 月のイメージはそのままにしておくことも、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ーでアクションを開き、「設定」タブを選択し、「ランタイム環境 docker image」ドロップダウンリストから「2024 年 3 月」のイメージを選択します。この選択により、アクションの YAML `Container` 定義にセクションが追加され、該当する 2024 年 3 月の画像が入力されます。
- 他のすべてのアクションは、ワークフローにいつ追加されたかにかかわらず、2022 年 11 月のイメージを使用します。現在、これらのアクションを 2024 年 3 月の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 月の画像は、から提供された最新の画像です。 CodeCatalystコンピュートタイプとフリートの組み合わせごとに 2024 年 3 月のイメージが 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ている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	CodeCatalyst Linux x86_64 用Amazon EC2-**CodeCatalystLinux_x86_64:2024_03**	CodeCatalyst リナックス x86_64 用 Lambda-**CodeCatalystLinuxLambda_x86_64:2024_03**	CodeCatalyst Linux Arm64 用Amazon EC2-**CodeCatalystLinux_Arm64:2024_03**	CodeCatalyst ナックス A Lambda-**CodeCatalystLinuxLambda_Arm64:2024_**
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	正しい (17)	17 が修正されました	17 が修正されました	17 が修正さ
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

コンピュートタイプとフリートの組み合わせごとに 1 つの 2022 年 11 月のイメージがあります。プロビジョニングされたフリートを設定している場合は、ビルドアクションで利用可能な 2022 年 11 月の Windows イメージもあります。

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	CodeCatalyst Linux x86_64 用Amazon EC2-**CodeCatalystLinux_x86_64:2022_11**	CodeCatalyst リナックス x86_64 用 Lambda-**CodeCatalystLinuxLambda_x86_64:2022_11**	CodeCatalyst Linux Arm64 用Amazon EC2-**CodeCatalystLinux_Arm64:2022_11**	CodeCatalyst ナックス A Lambda-**Co ystLinux ambda_Ar 64:2022_**
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	正しい (17)	17 が修正されました	17 が修正されました	17 が修正さ
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	CodeCatalyst Linux x86_64 用Amazon EC2-**CodeCatalystLinux_x86_64:2022_11**	CodeCatalyst リナックス x86_64 用 Lambda-**CodeCatalystLinuxLambda_x86_64:2022_11**	CodeCatalyst Linux Arm64 用Amazon EC2-**CodeCatalystLinux_Arm64:2022_11**	CodeCatalナックス A Lambda-**CystLinuxambda_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?

で提供されるアクティブイメージに、 CodeCatalyst 必要なツールが含まれていない場合は、次の 2 つの選択肢があります。

- 必要なツールを含むカスタムランタイム環境 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アクション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させることができます。

  たとえば、ビルドまたはテストアクションの YAML `Steps` コードのセクションに次の指示を含めることができます。

```
Configuration:
  Steps:
    - Run: ./setup-script
```

次に *setup-script* 命令は以下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。ビルドとテストアクションのリファレンス

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 Active イメージを使用したくない場合は CodeCatalyst、カスタムランタイム環境 Docker イメージを提供できます。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てください。イメージは Docker Hub、Amazon エラスティックコンテナレジストリ、または任意のパブリックリポジトリに配置できます。

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「アプリケーションのコンテナ化」を参照してください。

以下の手順に従って、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ます。イメージを指定したら、 CodeCatalyst アクションの開始時にそれをコンピュートプラットフォームにデプロイします。

> **Note**
>
> 次のアクションは、カスタムランタイム環境の Docker イメージをサポートしていません: AWS CloudFormation デプロイスタック、ECS へのデプロイ、GitHub アクション。カスタムランタイム環境の Docker イメージも Lambda コンピューティングタイプをサポートしていません。

Visual

ビジュアルエディター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/ CodeCatalyst で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. 下部付近で、以下のフィールドに入力します。

   ランタイム環境 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ーを指定します。有効な値を次に示します。

   - `CODECATALYST`(YAML エディター)

     CodeCatalyst イメージはレジストリに保存されます。
   - Docker Hub (ビジュアルエディター) または `DockerHub` (YAML エディター)

     イメージは Docker Hub イメージレジストリーに保存されます。
   - その他のレジストリー (ビジュアルエディター) または `Other` (YAML エディター)

     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ーならどれでも使用できます。
   - Amazon エラスティックコンテナレジストリ (ビジュアルエディタ) または `ECR` (YAML エディタ)

     イメージは Amazon エラスティックコンテナレジストリのイメージリポジトリに保存されます。Amazon ECR リポジトリ内のイメージを使用するには、このアクションが 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、以下の権限とカス

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。(必要に応じて、既存のロールを変更してアクセス権限とポリシーを含めることができます)。

IAM ロールのロールポリシーには、以下のアクセス権限が含まれている必要があります。

- ecr:BatchCheckLayerAvailability
- ecr:BatchGetImage
- ecr:GetAuthorizationToken
- ecr:GetDownloadUrlForLayer

IAM ロールには以下のカスタム信頼ポリシーが含まれている必要があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                 ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

IAM ロールの作成に関する詳細については、『IAM ユーザーガイド』の「カスタム信頼ポリシーを使用してロールを作成する (コンソール)」を参照してください。

ロールを作成したら、環境を通じ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レジストリーを使用している場合は、イメージを以下のアクティブ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 レジストリー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、カスタムレジストリが期待する値にイメージを設定します。

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. [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. 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. ランタイム環境の Docker イメージを割り当てたいアクションを探します。
6. アクションに、ContainerRegistryImageセクションと基になるプロパティを追加します。詳細については、の説明Container、RegistryImageアクションおよびアクションのプロパティを参照してください。
7. (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。
8. [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。

例

以下の例は、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内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
- 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 のサポートを追加しています。

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

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 のサポートを追加しています。

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
```

環境を使用する

開発環境 と混同されない環境 は、コードのデプロイ先です。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ています。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、 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 IAM AWS 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可能性がありますか？

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。

どのアクションが環境をサポートしていますか？

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。 「 AWS CloudFormation デプロイスタック」アクションの追加
- Amazon ECS へのデプロイ – 詳細については、「」を参照してください。 「Amazon ECS にデプロイ」アクションの追加
- Kubernetes クラスターへのデプロイ – 詳細については、「」を参照してください。 「Kubernetes クラスターへのデプロイ」アクションの追加
- AWS CDK デプロイ – 詳細については、「」を参照してください。 「AWS CDK デプロイ」アクションの追加

> **Note**
>
> AWS アクションがアカウント内のオペレーションにアクセスして実行できるようにする場合は、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、前述のアクションを含むが、これらに限定されません。どのアクションが環境の関連付けをサポートしているかは、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、どのAWSリージョンのリソースも表示できます。

環境は必須ですか？

環境は、それが割り当てられている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、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。

環境を作成する

以下の手順に従って、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、「セットアップ CodeCatalyst」を参照してください。
- CodeCatalyst プロジェクト。詳細については、「ブループリントを使っ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、「AWS アカウント スペースの管理」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アプリケーションが意図したとおりに動作していることを確認できます。
   - 本番稼働 – 公開され、確定済みのアプリケーションをホストする「本番稼働」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、実稼働環境にデプロイされているアクションをすばやく確認できます。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、「 CodeCatalyst 管理者ガイド」の「Amazon Virtual Private Cloud の管理」を参照してください。
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AWS アカウント スペースの管理。
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 ロールは 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の取得に

加えて、 アクションのデプロイ情報を環境ページにインポートすることもできます。詳細については、「どのアクションが環境をサポートしていますか？」を参照してください。

次の手順を使用して、環境、アカウント接続、IAM ロールをアクションに関連付けます。

ステップ 1: 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

次の手順を使用して、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、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

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、「どのアクションが環境をサポートしていますか？」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、環境を使用する「」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。では、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AWS アカウント スペースの管理。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

Amazon S3 や Amazon ECR などの AWSのサービスにアクセスして操作するために、このアクションが使用する 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。IAM ロールをアカウント接続に追加するには、「」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る場合があります。このロールの詳細については、「アカウントとスペース用の CodeCatalystWorkflowDevelopmentRole-***spaceName*** ロールの作成」を参照してください。CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールには、セキュリティ上の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、セキュリティ上の問題が少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。

リストにロールが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ことが原因です。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) コミットする前に、検証を選択してワークフローの 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て、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、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

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定義リファレンス「」を参照してください。

8. (オプション) コミットする前に、検証を選択してワークフローの 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て、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、およびロールをワークフローアクションに関連付けた後、環境ページにデプロイ情報を入力できます。次の手順を使用して、環境ページに値を入力します。

> **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、「どのアクションが環境をサポートしていますか？」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の実行が自動的に開始しなかった場合はステップ 1: 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、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、ステータスが SUCCEEDED になっているデプロイが表示されていることを確認します。これは、ワークフローの実行によってアプリケーションリソースが正常にデプロイされたことを示します。

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ていることを確認します。

環境の管理

次の手順に従って、環境を VPC 接続または に関連付けて管理しますAWS アカウント。

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従い、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。同じ VPC 接続を 1 つ以上の環境で使用できます。

次の手順を使用して、VPC 接続を環境と関連付けます。

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

次の手順を使用して、 を環境AWS アカウントと関連付けます。

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、 の関連付け を選択します。これにより、選択した AWS アカウントがこの環境に関連付けられます。

詳細については、「AWS アカウント スペースの管理」を参照してください。

ファイルキャッシュの操作

ファイルキャッシュが有効になっている場合、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、以降のワークフロー実行時にそのキャッシュから復元します。キャッシュにより、実行ごとに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で発生するレイテンシーが軽減されます。 CodeCatalyst また、フォールバックキャッシュもサポートしています。フォールバックキャッシュを使用すると、必要な依存関係の一部を含む部分キャッシュを復元できます。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 CodeCatalyst の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 コンピュートタイプを使用するように設定されている場合のみ使用できます。

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し、それぞれがプロパティの下で参照されます。FileCaching各キャッシュには、特定のパスで指定されたディレクトリが保存されます。指定したディレクトリは、future ワークフロー実行時に復元されます。以下は、とという名前の複数のキャッシュを含むキャッシュ用の YAML スニペットの例です。cacheKey1 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 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の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、一部のアクション実行では、リモートキャッシュのダウンロードによる遅延が発生する可能性があります。

CodeCatalyst キャッシュアクセス制限を適用して、あるワークフロー内のアクションが別のワークフローのキャッシュを変更できないようにします。これにより、ビルドやデプロイメントに影響する不正なデータをプッシュする可能性のある他のワークフローから各ワークフローを保護できます。キャッシュスコープでは、すべてのワークフローとブランチペアにキャッシュが分離されるため、制限が適用されます。たとえば、`workflow-A`feature-Aブランチには兄弟ブランチとは異なるファイルキャッシュがあります。`workflow-A` feature-B

キャッシュミスは、ワークフローが特定のファイルキャッシュを探しても見つからない場合に発生します。これは、新しいブランチが作成された場合や、新しいキャッシュが参照されたのに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も発生する可能性があります。デフォルトでは、キャッシュが最後に使用されてから 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、CodeCatalyst フォールバックキャッシュをサポートしています。フォールバックキャッシュは代替キャッシュであり、部分キャッシュを復元することができます。部分キャッシュは古いバージョンのキャッシュでもかまいません。キャッシュは、最初にプロパティ名と一致するものを検索して復元され、`FileCaching`見つからない場合は評価されます。`RestoreKeys`プロパティ名とすべてのキャッシュミスがあった場合でも`RestoreKeys`、キャッシュはベストエフォート型であり、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

次の手順に従って、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ーを使用してキャッシュを追加するには

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. Edit (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、「キャッシュを追加」を選択し、以下のように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のプロパティ名を指定します。キャッシュプロパティ名はワークフロー内で一意でなければなりません。各アクションには最大 5 つのエントリを含めることができますFileCaching。

   [Path] (パス)

   キャッシュの関連パスを指定してください。

   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の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。各キャッシュには最大 5 つのエントリを入れることができますRestoreKeys。
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. [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キャッシュを追加するには

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. Edit (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に、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. [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。

制約

RestoreKeysプロパティ名とには次のような制約があります。

- 名前はワークフロー内で一意でなければなりません。
- 名前は、英数字 (A ~ Z、a ~ z、0 ~ 9)、ハイフン (-)、およびアンダースコア (_) に制限されています。
- 名前には 180 文字まで使用できます。
- 各アクションには最大 5 FileCaching つのキャッシュを入れることができます。
- 各キャッシュには最大 5 RestoreKeys つのエントリを入れることができます。

パスには以下の制約があります。

- アスタリスク (*) は使用できません。
- パスには最大 255 文字まで使用できます。

パッケージを操作する

パッケージは、ソフトウェアをインストールして依存関係を解決するの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。 CodeCatalyst npm パッケージ形式をサポートします。

パッケージは以下のもので構成されます。

- 名前 (例えば、webpackはよく使われる npm パッケージの名前)
- オプションの名前空間 (例:in) @types @types/node
- バージョンセット (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)

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たり、CodeCatalyst パッケージリポジトリからパッケージを利用したりできます。 CodeCatalyst ビルドアクションまたはテストアクションをパッケージリポジトリで設定して、アクションの npm クライアントが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内のパッケージ CodeCatalyst。

> **Note**
>
> 現在、 CodeCatalyst ビルドアクションとテストアクションはパッケージリポジトリをサポートしています。

トピック

- CodeCatalyst ワークフローでのパッケージリポジトリの使用
- ワークフローでのパッケージの使用例

CodeCatalyst ワークフローでのパッケージリポジトリの使用

では CodeCatalyst、 CodeCatalyst ワークフロー内のビルドアクションとテストアクションに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構成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。

次の手順に従って、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ー)

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. ワークフロー図で、パッケージリポジトリで設定したい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を追加」を選択します。
11. Package リポジトリードロップダウンメニューで、 CodeCatalyst アクションで使用する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。Package 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したい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と、指定したパッケージリポジトリが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。そのスコープのパッケージが npm クライアントからリクエストされた場合、npm クライアントはデフォルトの代わりにそのリポジトリを使用します。各スコープ名の先頭には「@」を付ける必要があります。

    省略すると、指定し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。Scopes

    スコープについて詳しくは、「スコープ付きパッケージ」を参照してくださいPackage 名前空間。
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. [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

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 エディター)

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. アクションに、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、Packagesビルドとテストアクションのリファレンスアクションの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. [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。

ワークフローでのパッケージの使用例

以下の例は、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:でパッケージを定義する NpmConfiguration
- 例:デフォルトレジストリのオーバーライド

- 例:パッケージレジストリ内のスコープのオーバーライド

例:でパッケージを定義する **NpmConfiguration**

次の例は、NpmConfigurationワークフロー定義ファイル内で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構成しています。

```
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ーは、main-repoスコープなしで定義されているとおりに設定されています。@scope1スコープは PackageRegistries for で設定されますscoped-repo。

例:デフォルトレジストリのオーバーライド

次の例は、デフォルトレジストリをオーバーライド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構成しています。

```
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 が接続することになります。my-repo-3これはリポジトリとを上書きします。my-repo-1 my-repo-2

例:パッケージレジストリ内のスコープのオーバーライド

次の例は、パッケージレジストリ内のスコープをオーバーライド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構成しています。

```
default: my-default-repo
@scope1: my-repo-1
@scope2: my-repo-2
```

オーバーライドするスコープを含めると、最後のリポジトリが優先されます。この例では、@scope2スコープが最後に設定されるのは for PackageRegistries です。my-repo-2これにより、@scope2に設定されたスコープがオーバーライドされます。my-repo-1

ランの処理

実行とは、ワークフローを 1 回繰り返すことです。実行中、 CodeCatalystワークフロー設定ファイルに定義されている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。

トピック

- ワークフロー実行の開始
- ワークフロー実行の停止
- ワークフロー実行のステータスと詳細の表示
- キュー実行、代替実行、parallel 実行の設定

ワークフロー実行の開始

以下の手順を使用して、ワークフローを手動で開始します。

> **Note**
>
> トリガーを設定することで、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。

ワークフローを開始するには、手動で実行します。

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. 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. [実行する] を選択します。

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の実行を停止します。実行が誤って開始された場合は、その実行を停止したほうがよい場合があります。

ワークフローの実行を停止すると、 CodeCatalyst 実行中のアクションが完了するのを待ってから、コンソールで実行が 「停止済み」とマークされます CodeCatalyst 。開始する機会がなかったアクションは開始されず、「放棄」とマークされます。

> **Note**
>
> 実行がキューに入っている (つまり、実行中のアクションがない) 場合、その実行はただちに停止されます。

ワークフローの実行を停止するには

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」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ステータスと詳細の表示

1 つ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。

> **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

- ワークフロー実行ステータス
- 1 回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実行のステータスと詳細の表示
- 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
- ワークフロー図でのワークフローの実行の表示

ワークフロー実行ステータス

ワークフロー実行には、次のいずれかのステータスがあります。

- 成功 – ワークフローの実行は正常に処理されました。
- 失敗 – 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。
- 停止 — 進行中にワークフローの実行を停止したユーザー。
- 停止中 — ワークフロー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、ワークフロー実行は 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 を設定した場合にのみ発生します。ワークフロー実行は、後続のワークフロー実行がそれを置き換えた CodeCatalyst ため、 によってキャンセルされました。

1 回の実行のステータスと詳細の表示

1 つのワークフロー実行のステータスと詳細を表示して、成功したかどうかの確認、完了時刻の確認、または誰が、何が開始したかの確認を行うことができます。

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、「 を実行」を選択します。
6. 実行履歴 で、実行 ID 列で実行を選択します。例えば、「Run-95a4d」と入力します。
7. 実行の名前で、次のいずれかを実行します。

   - ビジュアルで、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確認できます (「」を参照ワークフロー実行ステータス)。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、実行中に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を表示する「」または「」を参照してくださいデプロイログを表示する。
   - YAML は、実行に使用されたワークフロー定義ファイルを表示します。
   - 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確認するためのアーティファクト。アーティファクトの詳細については、「アーティファクトによる作業」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、「テストレポートタイプ」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変数の操作。

> **Note**
>
> 実行の親ワークフローが削除された場合、実行の詳細ページの上部にこの事実を示すメッセージが表示されます。

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、プロジェクトでどの程度のワークフローアクティビティが実行されているかを把握し、ワークフローの全体的な状態について確認することもできます。

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にある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。実行ステータスの詳細については、「」を参照してくださいワークフロー実行ステータス。
   - トリガー – ワークフローの実行を開始したユーザー、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、「トリガーの使用」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「利用不可」に設定されている場合、通常は関連する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 – ワークフローの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、ワークフロー内でボトルネックが発生している実行がないか確認したり、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、「 を実行」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 (進行中でない) 実行を表示します。

     実行ステータスの詳細については、「」を参照してくださいワークフロー実行ステータス。

ワークフロー図でのワークフローの実行の表示

ワークフローのすべての実行のステータスは、ワークフローを通じて進行するにつれて表示できます。実行はワークフロー図内に表示されます (リストビューではなく) 。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確認できます。

ワークフローを通じて複数の実行のステータスを一緒に表示するには

> **Note**
>
> この手順は、ワークフローがキューまたは置換さ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、「キュー実行、代替実行、parallel 実行の設定」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参照し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、最後に終了した実行を示しています。具体的には :

   - 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、開始を待ちます。
   - アクション間に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ちます。
   - アクション内に表示される実行は、1 です。現在、 アクションで処理されています。2. アクションで処理が終了しました。または、3. は アクションで処理されませんでした (通常は、前のアクションが失敗したためです) 。

キュー実行、代替実行、parallel 実行の設定

デフォルトでは、複数のワークフローが同時に実行されると、 CodeCatalyst それらをキューに入れ、開始された順序で 1 つずつ処理します。この既定の動作は、実行モードを指定することで変更できます。実行モードはいくつかあります。

- (デフォルト) キュー実行モード — CodeCatalyst プロセスが 1 つずつ実行されます。
- 優先実行モード — CodeCatalyst プロセスが 1 つずつ実行され、新しい実行が古いプロセスを追い越します。
- parallel 実行モード — CodeCatalyst プロセスを並列に実行

トピック

キュー実行モードについて

キュー実行モードでは、実行が連続して行われ、待機中の実行がキューを形成します。

キューはアクションとアクショングループへのエントリポイントで形成されるため、同じワークフロー内に複数のキューを設定できます (を参照)。Figure 1キューに入っているランがアクションに入ると、そのアクションはロックされ、他のランは入力できなくなります。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。

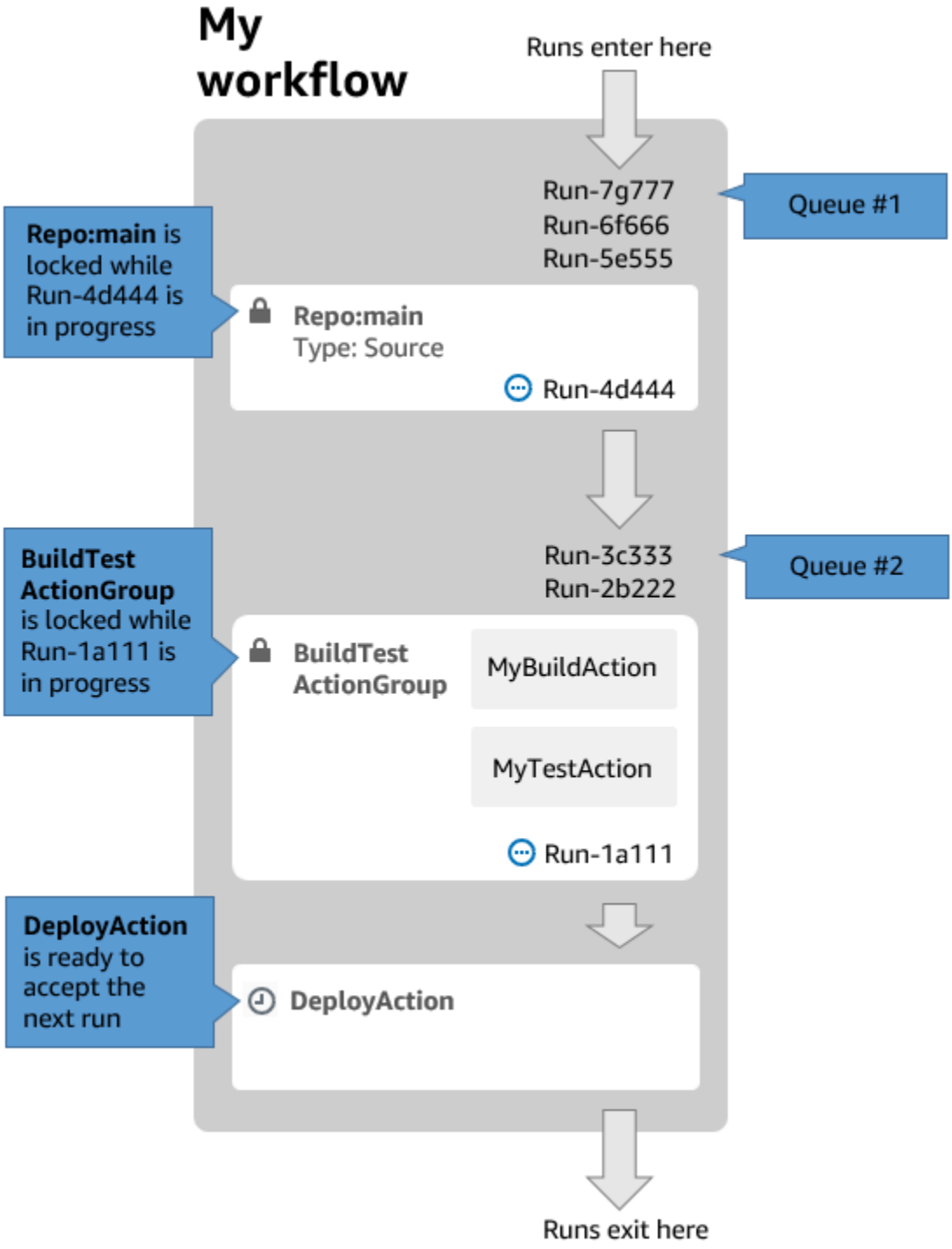
- 7 回実行してワークフローを進めています。
- 2 つのキュー。1 つは入力ソース (repo: Main) のエントリの外側にあり、もう 1 つはアクションのエントリの外側にあります。BuildTestActionGroup
- ロックされている 2 つのブロック:入力ソース (repo: Main) と. BuildTestActionGroup

ワークフローが実行されて処理が終了すると、次のようになります。

- Run-4D444 がソースリポジトリのクローニングを完了すると、入力ソースを終了し、Run-3C333 の背後にあるキューに加わります。次に、Run-5E555 が入力ソースに入ります。

- Run-1a111 はビルドとテストを終了すると、アクションを終了してアクションを開始します。BuildTestActionGroupDeployAction次に、Run-2B222 がアクションに入ります。BuildTestActionGroup

図 1:「キュー実行モード」で構成されたワークフロー



以下の場合はキュー実行モードを使用してください。

- one-to-one 機能と実行の関係を維持したい場合、置き換えモードを使用するとこれらの機能をグループ化できます。たとえば、コミット 1 に機能 1 をマージすると実行 1 が開始され、コミット 2 の機能 2 をマージすると、実行 2 が開始される、という具合になります。キューモードではなく置き換えモードを使用すると、機能 (とコミット) は実行時にグループ化され、他の機能よりも優先されます。
- parallel モードの使用時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する必要があります。たとえば、Wang と Saanvi という 2 人のソフトウェア開発者が、Amazon ECS クラスターにデプロイするワークフロー実行をほぼ同時に開始した場合、Wang の実行ではクラスターの統合テストが開始され、Saanvi の実行では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かもしれません。その場合、2 回の実行でお互いの変更が予期せぬ方法で上書きされる可能性があります。
- CodeCatalyst 実行を処理するために使用するコンピュートリソースの負荷を制限する必要があります。たとえば、ワークフローに 3 つのアクションがある場合、同時に実行できるのは最大 3 回です。一度に実行できる実行回数に制限を設けると、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のサービスに送信されるリクエストの数を制限する必要があります。たとえば、ワークフローに Docker Hub からイメージを取得する命令を含むビルドアクションがあるとします。Docker Hub では、1 時間あたりに実行できるプルリクエストの数がアカウントごとに特定の数に制限されており、制限を超えるとロックアウトされます。キュー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 時間あたりに生成される Docker Hub へのリクエストの数が少なくなるため、ロックアウトやその結果生じるビルドや実行の失敗の可能性が制限されます。

キューの最大サイズ:50

最大キューサイズに関する注意事項:

- 最大キューサイズとは、ワークフロー内のすべてのキューで許可される最大実行回数のことです。
- キューの実行数が 50 回を超えると、51 CodeCatalyst 回目以降の実行がドロップされます。

障害時の動作:

アクションによる処理中に実行が応答しなくなった場合、その背後の実行はアクションがタイムアウトにな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、そのアクションの背後にキューに入れられた最初の実行が続行できます。

置き換えられた実行モードについて

代替実行モードはキュー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、後の実行が前の実行に取って代わり (引き継ぎ)、前の実行はキャンセルされ、「置き換え済み」とマークされます。
- 最初のbullet で説明した動作の結果として、置き換えられた実行モードが使用されている場合、キューには 1 つの実行しか含めることができません。

Figure 1のワークフローをガイドとして使用すると、置き換えられた実行モードをこのワークフローに適用すると、次のような結果になります。

- 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 に取って代わり、キュー #2 に残っている唯一のランになります。RUN-2B222 はキャンセルされます。

必要であれば、置き換えられた実行モードを使用してください。

- キューモードよりもスループットが良い
- サードパーティのサービスへのリクエストはキューモードよりもさらに少なくなります。これは、Docker Hub のようにサードパーティのサービスにレート制限がある場合に有利です。

parallel 実行モードについて

parallel 実行モードでは、実行は互いに独立しており、開始する前に他の実行が完了するのを待つ必要はありません。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の時間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、parallel 実行モードを使用してください。

> **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、parallel モードを使用しないでください。parallel ワークフローの実行で共有リソースを同時に変更しようとすると、競合状態が発生し、予期しない結果につながります。

最大parallel 実行数:1 CodeCatalyst スペースあたり 1000

実行モードの変更

実行モードは、キュー、置き換え、またはparallel に設定できます。デフォルトは「キューに入っている」です。

実行モードをキューまたは置換からparallel に変更すると、 CodeCatalyst キューに入っている実行がキャンセルされ、現在アクションによって処理されている実行がキャンセルされる前に終了できるようになります。

実行モードをparallel からキューまたは置き換え済みに変更すると、現在実行中のすべてのparallel CodeCatalyst 実行が完了します。実行モードをキューまたは置き換えモードに変更した後に開始した実行には、新しいモードが使用されます。

Visual

ビジュアルエディターを使用して実行モ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. 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. 右上の [ワークフロープロパティ] を選択します。
7. [詳細設定] を展開し、[実行モード] で次のいずれかを選択します。

   a. 待機中 —「」を参照 キュー実行モードについて

b. 置き換え済み —「」を参照 置き換えられた実行モードについて

c. パラレル —「」を参照 parallel 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. [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実行モ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. 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以下のようにプロパティを追加します。

```
Name: Workflow_6d39
SchemaVersion: "1.0"
RunMode: QUEUED|SUPERSEDED|PARALLEL
```

詳細については、RunModeトップレベルのプロパティのセクションにあるプロパティの説明を参照してくださいワークフロー定義リファレンス。

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. [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。

シークレットの使用

ワークフローで認証情報などの機密データを使用しなければならない場合があります。シークレットを含むリポジトリにアクセスできるユーザーは誰でも表示できるため、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。では 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。1 つのアクションにつき最大 5 つのシークレットを持つことができます。

> **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、次にシークレット 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、次にシークレット 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、ビューから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

- ワークフロー内のシークレットの参照

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、次の手順に従います。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、次にシークレット 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>}
```

ワークフロー内のシークレットの参照

ワークフローで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) コミットする前に、検証を選択してワークフローの 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て、もう一度コミットを選択します。

シークレットの削除

次の手順を使用して、シークレットとシークレット参照識別子を削除します。

> **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、次回実行時にアクションが失敗します。

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) コミットする前に、検証を選択してワークフローの 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て、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、次にシークレット 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. 削除をクリックします。

5. **delete** を入力して削除を確認します。
6. [削除] をクリックします。

ソースの操作

ソースは入力ソースとも呼ばれ、ワークフローアクションがタスクを実行するためにアクセスする必要のあるソースリポジトリです。たとえば、ワークフローアクションは、単体テストを取得してアプリケーションのソースファイルに対して実行するために、ソースにアクセスする必要がある場合があります。

CodeCatalyst ワークフローは以下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のソースリポジトリ CodeCatalyst。
- GitHub ソースリポジトリ — 詳細については、を参照してください。 GitHub でのリポジトリの使用 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を指定する。
- 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た変数

ワークフロー定義ファイルを保存するソースの指定

以下の手順に従って、 CodeCatalyst 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します。 GitHub ソースリポジトリを指定したい場合は、代わりにを参照してください GitHub でのリポジトリの使用 CodeCatalyst。

ワークフロー定義ファイルが置かれているソースリポジトリは、`WorkflowSource`というラベルで識別されます。

> **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置かれてい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。最初のコミットの後にリポジトリを変更する唯一の方法は、別のリポジトリに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/ 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ー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。

   ワークフローの作成プロセス中に、 CodeCatalyst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よう求められます。

ワークフローアクションが使用するソースを指定する。

以下の手順に従って、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の Docker イメージにアーティファクトをダウンロードして、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。ソースリポジトリとは、ワークフロー定義ファイルが (ディレクトリ内の) 存在するソースリポジトリです。`.codecatalyst/workflows/WorkflowSource`このソースリポジトリはラベルで表されます。

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ー)

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. 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、以下の操作を行います。

   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`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すラベルだけです。

   ソースを省略する場合は、その下に少なくとも 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、「ソースの操作」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. [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

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 エディター)

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. アクションに、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、Sourcesワークフロー定義リファレンスアクションの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. [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。

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存在するファイルがあり、それらのファイルをワークフローアクションの 1 つで参照する必要がある場合は、以下の手順を実行してください。

> **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に、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、my-appWorkflowSourcefile1.jarアクションはソースリポジトリのルートにあるディレクトリを検索して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た変数

ワークフローを実行すると、そのソースによって変数が生成され、後続のワークフローアクションで使用できる変数が生成されます。詳細については、"BranchName" と "CommitId" 変数のを参照してください定義済み変数のリスト。

トリガーの使用

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をなくすことができます。

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の実行を開始します。
- 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の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次の午前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表示するには、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、「ブループリントを使っ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確定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mainときに機能ブランチを 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。

  このプルリクエストでは、ソフトウェア開発者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、すべてが正常に見えることを確認します。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ビルド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 (つまり、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ビルド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になります。 mainは、ブランチのコード (プルリクエストのコードが含まれるようになりました) 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ビルド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。

トリガーの例と追加の説明については、「」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。送信元 (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 へのプッシュによってワークフローがmainアクティブ化されます。

  ```
  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によってワークフローがmainアクティブ化されます。

  ```
  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送信元ブランチで実行されます (トリガー設定で送信先ブランチを指定した場合でも) 。
- 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について考えてみましょう。

  ```
  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、 の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てください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

次の手順を使用して、プッシュ、プル、またはスケジュールの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. 「トリガーを追加」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の実行を開始します。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) のファイルが使用されます。

   - 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開いたり、更新されたり、閉じたりし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) のファイルが使用されます。

   - 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 個以下のワークフローを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、実行の間に十分な時間で設定してください。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ている (ビジュアルエディタ) または OPEN (YAML エディタ)

  ワークフロー実行は、プルリクエストが作成されたときに開始されます。
- 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、プルリクエストが閉じら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、例から最も理解できます。詳細については、「例: プル、ブランチ、「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) に対して新しいリビジョンが作成されました REVISION

  ワークフローの実行は、プルリクエストのリビジョンが作成されたときに開始されます。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に REVISIONイベントを含めると、REVISIONは のスーパーセットであるため、 OPENイベントを省略できます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のみ表示されます。

スケジュールされたワークフローの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 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ています。

*##*時間*days-of-month#days-of-week#*

cron 式の例

分	時間	日	月	曜日	年	意味
0	0	?	*	MON-FRI	*	毎週月曜日から金曜日まで午前 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	日	月	曜日	年	意味
0/30	22-2	?	*	土 - 日	*	開始日の午後 10 時から翌日の午前 2 時 (UTC +0) の間、土曜日から日曜日まで 30 分ごとにワークフローを実行します。
45	13	L	*	?	2023-2027	2023 年から 2027 年までの月の最後の日の午後 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、必ず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。

- 実行間隔を 30 分以上設定します。より速い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。フィールドのいずれかで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 機能します。

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)

ワークフローの実行を開始するタイミングを確認するために、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、 を使用して、 で始まるすべてのブランチ`main.*`を照合しますmain。

指定するブランチは、トリガーのタイプによって異なります。

- プッシュトリガーには、 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。一致したブランチ内のファイルを使用して、一致したブランチごとに 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には、 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 つのワークフロー実行が開始されます。

  例: `main.*`、`mainline`、 `v1\-.*` (で始まるブランチと一致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。ワークフロー定義ファイルと、一致したブランチ内のソースファイルを使用して、一致したブランチごとに 1 つのワークフロー実行が開始されます。

例: `main.*`、`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 (プッシュトリガー用) 。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 つ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。

ワークフローの実行を開始するタイミングを確認するために、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照合できます。

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) コミットする前に、検証を選択してワークフローの 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て、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、ワークフロー定義リファレンス 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) コミットする前に、検証を選択してワークフローの 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て、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持つトリガー
- 例: スケジュール、プッシュ、ブランチを含むトリガー
- 例: プルとブランチを持つトリガー
- 例: プル、ブランチ、「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、コードがソースリポジトリ内の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) 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 定義ファイルと他のソースファイルを使用してワークフローの実行 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 定義ファイルと他のソースファイルを使用してワークフローの実行 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を実行する場合は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main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
    Branches:
      - main
```

例: シンプルなプルリクエストトリガー

次の例は、ソースリポジトリでプルリクエストが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。

このトリガーを有効にすると、 はワークフロー定義ファイルと、プル元のブランチ (つまりソースブランチ) 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、 という送信元ブランチfeature-123と という送信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) にワークフローの実行を開始するトリガーを示しています。

このトリガーを有効にすると、 は、このトリガーを含む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、、、 `main` `release-v1`がありfeature-123、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 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 のファイルを使用し`release-v1`、もう 1 つは のファイルを使用します`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持つトリガー

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 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て`main`、もう 1 つは のファイルを使用して実行します`release-v1`。ブランチbugfix-1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、およびコードがmainブランチにプッシュされるたびにワークフローの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。ワークフローの実行では、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。ワークフローの実行では、ワークフロー定義ファイルと、送信先ブランチ () にあるそ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: プルとブランチを持つトリガー

次の例は、 という送信先ブランチを使用してプルリクエストを開いたり変更したり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、ワークフロー定義ファイルと、ソースブランチ内の他のソースファイル (からプルするブランチ) が使用されます。

```
Triggers:
  - Type: PULLREQUEST
    Branches:
      - main
    Events:
```

```
      - OPEN
      - REVISION
```

例: プル、ブランチ、「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を使用して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、 (現在クローズされている) 送信元ブランチ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、マージせずにプルリクエストを閉じることです。

```
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、 filename.txt ファイル、または src ディレクトリ内のファイルが mainブランチで変更されるたびにワークフローの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\/.*
```

変数の操作

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。

ワークフローでは次の 2 種類の変数を使用できます。

- 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。
- 定義済みの変数 — ワークフローによって自動的に生成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 また、変数のように動作し、GitHub 他のアクションで参照できる出力パラメータもサポートしています。詳細については、「 GitHub アクション出力パラメータの操作」を参照してください。

トピック

ユーザー定義変数を使用する

ユーザー定義変数は、ユーザーが定義するキーと値のペアです。次の 2 つのタイプがあります。

- プレーンテキスト変数、または単純変数 — 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 — 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、値にはプライベートにしておきたい情報が含まれています。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てください。

> **Note**
>
> このガイドでは、簡潔に説明するために、「変数」という用語を平文の変数を意味して使用しています。

トピック

変数を定義する。

変数は次の 2 つの方法で定義できます。

- Inputsワークフローアクションのセクション — 「Inputs」セクションの「変数を定義するには」を参照してください。
- Stepsワークフローアクションのセクションでは、「「ステップ」セクションで変数を定義するにはを参照してください。

> **Note**
>
> Steps CodeCatalyst このメソッドはビルドアクション、GitHub テストアクション、アクションアクションでのみ機能します。Stepsセクションを含むアクションはこれらだけだからです。

例については、「例」を参照してください。

Visual

「Inputs」セクション (ビジュアルエディター) で変数を定義するには

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. 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する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 (-)、およびアンダースコア (_) に制限されています。スペースは使用できません。引用符を使用して変数名に特殊文字やスペースを含め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変数の操作。
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. [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

「Inputs」セクション (YAML エディター) で変数を定義するには

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. ワークフローアクションに、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、「例」を参照してください。詳細については、該当するアクションの ワークフロー定義リファレンス「」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. [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。

Visual

「Steps」セクション (ビジュアルエディター) で変数を定義するには

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. 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. GitHubシェルコマンドまたはアクション YAML (使用可能な場合) で、アクション内の変数を明示的または暗黙的に定義します。`Steps`

   - 変数を明示的に定義するには、bash コマンドにその変数をセクションに直接含めます。`Steps`
   - 変数を暗黙的に定義するには、`Steps`アクションのセクションで参照されるファイルで変数を指定します。

     例については、「例」を参照してください。詳細については、アクションの ワークフロー定義リファレンス「」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. [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

「Steps」セクション (YAML エディター) で変数を定義するには

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. ワークフローアクションでは、`Steps`アクションのセクションで変数を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、bash コマンドにその変数をセクションに直接含めます。`Steps`
   - 変数を暗黙的に定義するには、`Steps`アクションのセクションで参照されるファイルで変数を指定します。

   例については、「例」を参照してください。詳細については、アクションの ワークフロー定義リファレンス「」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. [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。

シークレットの定義

CodeCatalystシークレットはコンソールの Secrets ページで定義します。詳細については、「シークレットの使用」を参照してください。

たとえば、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**

シークレットを定義したら、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。

変数をエクスポートして他のアクションでも使用できるようにする。

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、その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。使用している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ーの説明を実行し、アクションの「出力」タブに「変数」ボタンがあるかどうかを確認してください。「はい」の場合、変数のエクスポートはサポートされています。
- 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 GitHub 出力パラメータをエクスポートして他のアクションでも使用できるようにする。。

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てください。詳細については、「変数を定義する。」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ー)

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. 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、`InputsSteps`同じアクションのまたはセクションですでに定義されている必要があります。
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. [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

変数をエクスポートするには (YAML エディター)

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. 変数をエクスポートするアクションに、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、「例」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. [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。

変数を定義するアクション内の変数を参照する

以下の説明を使用して、変数を定義するアクション内の変数を参照します。

> **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出力パラメータを参照する。。

前提条件

参照する変数が定義されていることを確認してください。詳細については、「変数を定義する。」を参照してください。

Visual

使用できません。YAML を選択すると YAML のインストラクションが表示されます。

YAML

変数を定義するアクション内の変数を参照するには

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. CodeCatalyst 参照する変数を定義するアクションで、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、「例」を参照してください。詳細については、にあるアクションのリファレンス情報を参照してください。ワークフロー定義リファレンス

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. [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。

別のアクションによって出力された変数を参照する。

以下の手順を使用して、他のアクションが出力する変数を参照します。

> **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出力パラメータを参照する。。

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てください。詳細については、「変数をエクスポートして他のアクションでも使用できるようにする。」を参照してください。

Visual

使用できません。YAML を選択すると YAML のインストラクションが表示されます。

YAML

別のアクション (YAML エディター) によって出力された変数を参照するには

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. CodeCatalyst アクションで、以下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を指定します。

> **Note**
>
> *action-group-name*アクショングループがない場合や、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省略できます。

- *action-name* と変数を出力するアクションの名前。
- *#########*。

例:

```
MySecondAction:
    Configuration:
      Steps:
        - Run: ${MyFirstAction.TIMESTAMP}
```

その他の例については、「例」を参照してください。詳細については、ワークフロー定義リファレンスアクションの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. [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。

シークレットを参照する

ワークフロー定義ファイル内の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。

例

次の例は、ワークフロー定義ファイル内の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て変数を定義する
- 例:Steps プロパティを使用して変数を定義する
- 例:Outputs プロパティを使用して変数をエクスポートする
- 例:同じアクションで定義された変数の参照
- 例:別のアクションで定義された変数を参照する
- 例:シークレットの参照

例:Inputs プロパティを使用して変数を定義する

次の例はVAR2、VAR1と 2 Inputs つの変数をセクションで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て変数を定義する

次の例は、DATEStepsセクション内の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て変数をエクスポートする

次の例は、と 2 REPOSITORY-URI つの変数を定義しTIMESTAMP、Outputsセクションを使用してそれらを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の参照

次の例はMyBuildAction、VAR1で変数を指定し、$VAR1を使用して同じアクションでその変数を参照する方法を示しています。

```
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を参照する

次の例はBuildActionA、TIMESTAMPで変数を指定し、Outputsプロパティを使用して工クスポートし、BuildActionB${BuildActionA.TIMESTAMP}を使用して参照する方法を示しています。

```
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はシークレットの鍵です。このシークレットのキーとそれに対応するパスワード値は、 CodeCatalyst ワークフロー定義ファイルで使用する前に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、「シークレットの使用」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
```

定義済みの変数を使用する

定義済み変数は、ワークフローによって自動的に生成され、ワークフローアクションで使用できるようになるキーと値のペアです。

定義済みの変数はどのワークフローアクションでも使用できます。

トピック

- 定義済み変数のリスト
- 定義済みの変数を参照する。
- ワークフローが出力する定義済み変数を決定する
- 例

定義済み変数のリスト

によって自動的に生成される定義済み変数については、以下のセクションを参照してください CodeCatalyst。

> **Note**
>
> このリストには、 CodeCatalyst CodeCatalyst ワークフローとアクションによって生成される定義済みの変数のみが含まれています。Actions や CodeCatalyst Labs アクションなど、GitHub 他の種類の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定義済み変数を決定する

リスト

- "BranchName" と "CommitId" 変数
- アクション変数をビルドしてテストします。
- 「Amazon S3 パブリッシュ」アクション変数
- 「AWS CDKブートストラップ」アクション変数
- AWS CDKアクション変数を「デプロイ」します。
- 「AWS Lambdainvoke」アクション変数
- 「AWS CloudFormationデプロイスタック」アクション変数
- 「Amazon ECS にデプロイ」アクション変数
- 「Kubernetes クラスターにデプロイ」アクション変数
- 「Amazon ECS タスク定義をレンダリング」アクション変数
- 「GitHub アクション」アクション変数

"BranchName" と "CommitId" 変数

CodeCatalyst ワークフローの実行時に BranchName "" と CommitId "" の変数を生成します。これらの変数について詳しくは、次の表を参照してください。

キー	値
CommitId	ワークフロー実行開始時のリポジトリの状態を表すコミット ID。 例: `example3819261db00a3ab59468c8b` 「例:「CommitId" 定義済み変数の参照」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:「BranchName" 定義済み変数の参照」も参照してください。

アクション変数をビルドしてテストします。

ビルドアクションとテストアクションは変数を自動的に生成しません。

「Amazon S3 パブリッシュ」アクション変数

Amazon S3 のパブリッシュアクションでは、変数は自動的に生成されません。

「AWS CDKブートストラップ」アクション変数

AWS CDKブートストラップアクションは、以下の定義済み変数を生成します。

キー	値
デプロイメントプラットフォーム	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`

キー	値
region	AWS リージョンAWS CDKワークフローの実行中にブートストラップスタックがデプロイされたのリージョンコード。 例: `us-west-2`
stack-id	AWS CDKデプロイされた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`
デプロイメントを省略	値がの場合、`true`AWS CDKワークフローの実行中に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。 この変数は値がの場合にのみ生成されます。`true` にハードコードされています。`true`

AWS CDKアクション変数を「デプロイ」します。

AWS CDKdeploy アクションは以下の変数を生成します。

キー	値
スタック ID	AWS CDKワークフローの実行中にデプロイされたアプリケーションスタックの Amazon リソースネーム (ARN)。

キー	値
	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。 にハードコードされています。`AWS:CloudFormation`
region	AWS リージョンワークフローの実行中にデプロイされたのリージョンコード。 例: `us-west-2`
デプロイメントを省略	の値は、`true`AWS CDKワークフローの実行中に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。 この変数は値がの場合にのみ生成されます。`true` にハードコードされています。`true`

キー	値
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出力変数をワークフロー変数として公開し、後続のワークフローアクションで使用できるようにします。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) CloudFormation の変数のみを公開します。どの変数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 回実行してから、実行詳細ページの「変数」タブを調べます。Variables タブにリストされている変数が希望どおりでない場合は、`CfnOutputVariables` YAML プロパティを使用して別の変数を設定できます。詳細については、CfnOutputVariablesのプロパティの説明を参照してください。「AWS CDKデプロイ」アクションリファレンス

「AWS Lambdainvoke」アクション変数

デフォルトでは、AWS Lambdainvoke 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。

たとえば、レスポンスペイロードが次のようになっているとします。

```
responsePayload = {
  "name": "Saanvi",
  "location": "Seattle",
  "department": {
    "company": "Amazon",
    "team": "AWS"
  }
}
```

... その場合、アクションは次の変数を生成します。

キー	値
名前	Saanvi
ロケーション	Seattle
department	{"会社」:「Amazon」、「チーム」: ""AWS}

> **Note**
>
> `ResponseFilters`YAML プロパティを使用して、どの変数を生成するかを変更できます。詳細については、AWS Lambda 「呼び出し」アクションリファレンス の「ResponseFilters」を参照してください。

「AWS CloudFormationデプロイスタック」アクション変数

Deploy AWS CloudFormation Stack アクションは以下の変数を生成します。

キー	値
デプロイメントプラットフォーム	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	AWS リージョンワークフローの実行中にデプロイ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`

「Amazon ECS にデプロイ」アクション変数

Amazon ECS へのデプロイアクションは、以下の変数を生成します。

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`
サービス	ワークフローの実行中にデプロイされた 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` `:8`前の例の「」は、登録されたリビジョンを示しています。
デプロイメント URL	Amazon ECS コンソールの [イベント] タブへのリンク。このタブでは、ワークフローの実行に関連する 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	AWS リージョンワークフローの実行中にデプロイされたのリージョンコード。 例: `us-west-2`

「Kubernetes クラスターにデプロイ」アクション変数

Kubernetes へのデプロイ (デプロイ) クラスターアクションでは以下の変数が生成されます。

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 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ています。`AWS:EKS`
メタデータ	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`
リソース	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。

キー	値
server	などの管理ツールを使用してクラスターと通信するために使用できる API サーバーエンドポイントの名前。kubectl 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`

「Amazon ECS タスク定義をレンダリング」アクション変数

Amazon ECS タスク定義のレンダリングアクションは、以下の変数を生成します。

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`

「GitHub アクション」アクション変数

「GitHub Actions」アクションは変数を自動的に生成しません。

定義済みの変数を参照する。

定義済みの変数を参照するには、以下の手順に従います。

前提条件

参照する定義済み変数の名前 (など) CommitId を決定します。詳細については、「ワークフローが出力する定義済み変数を決定する」を参照してください。

Visual

使用できません。YAML を選択すると YAML のインストラクションが表示されます。

YAML

定義済みの変数を参照するには (YAML エディター)

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. CodeCatalyst アクションでは、以下の構文を使用して定義済みの変数参照を追加します。

```
${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と一緒に。

> **Note**
>
> *action-group-name*アクショングループがない場合や、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省略できます。

- *action-name-or-WorkflowSource #:*

  変数を出力するアクションの名前。

  または

  WorkflowSource、BranchNameCommitId変数がまたは変数の場合。

- *#########*。

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、「例」を参照してください。詳細については、アクションの「」を参照してくださいワークフロー定義リファレンス。

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. [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。

ワークフローが出力する定義済み変数を決定する

ワークフローがどの定義済み変数を出力するかは、次の 2 つの方法で決定できます。

- ワークフローは 1 回実行してください。実行が完了すると、ワークフローによって生成された変数が実行詳細ページの「変数」タブに表示されます。詳細については、「ワークフロー実行の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。定義済み変数のリストこのリファレンスには、定義済みの各変数の変数名 (キー) と値が記載されています。

> **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のワークフローのクォータ CodeCatalyst。合計サイズが最大値を超えると、最大サイズに達した後に実行されるアクションは失敗する可能性があります。

例

以下の例は、ワークフロー定義ファイル内の定義済み変数を参照する方法を示しています。

例

- 例:「CommitId" 定義済み変数の参照
- 例:「BranchName" 定義済み変数の参照

例:「CommitId" 定義済み変数の参照

次の例は、CommitIdMyBuildActionアクション内で定義済みの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。

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、CommitIdどのアクションでも使用できます。

```
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:「BranchName" 定義済み変数の参照

次の例は、BranchNameCDKDeployアクション内で定義済みの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。

AWS CDKこの例では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、BranchNameどのアクションでも使用できます。

```
CDKDeploy:
    Identifier: aws/cdk-deploy@v1
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      StackName: app-stack-${WorkflowSource.BranchName}
```

ワークフロー定義リファレンス

以下は、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です。

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。このファイルは、~/.codecatalyst/workflows/ソースリポジトリのルートにあるフォルダーに保存されます。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して編集するには、vim などのエディターを使用するか、CodeCatalyst コンソールのビジュアルエディターや YAML エディターを使用できます。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**
>
> 以下の YAML プロパティのほとんどは、ビジュアルエディターに対応する UI 要素を備えています。UI 要素を検索するには、Ctrl+F を使用します。 要素は関連する 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
- トップレベルのプロパティ
- ビルドとテストアクションのリファレンス
- 「Amazon S3 公開」アクションリファレンス

- AWS CDK 「ブートストラップ」アクションリファレンス
- 「AWS CDK デプロイ」アクションリファレンス
- AWS Lambda 「呼び出し」アクションリファレンス
- AWS CloudFormation 「スタックのデプロイ」アクションリファレンス
- 「Amazon ECS へのデプロイ」アクションリファレンス
- 「Kubernetes クラスターへのデプロイ」アクションリファレンス
- GitHub 「アクション」アクションリファレンス
- 「Amazon ECS タスク定義をレンダリングする」アクションリファレンス

ワークフロー定義ファイルの例

以下は簡単なワークフロー定義ファイルの例です。これには、最上位のプロパティ、Triggersセクション、および 2 Actions Build つのアクションを含むセクションがいくつか含まれています。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規則と、このリファレンスドキュメントで使用されてい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 仕様に準拠しているため、その仕様で許可されていることはすべてワークフロー 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で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記載されている大文字と小文字を必ず使用してください。
- 特殊文字:,,, &,,,,, <, >,,, >{,,,,,,,,,},[,,],,,*,#,,?,|,-,,,,, = および ！ % @ : ` ,

  引用符を含めないと、前述の特殊文字が予期しない方法で解釈される可能性があります。
- プロパティ名:プロパティ名は (プロパティ値とは対照的に) 英数字 (a-z、A-Z、0-9)、ハイフン (-)、およびアンダースコア (_) に制限されています。スペースは使用できません。引用符や二重引用符を使用して、プロパティ名に特殊文字やスペースを含めることはできません。

  使用不可:

  ```
  '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 (たとえば、、\t) \n 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。
  - エスケープコードを文字列として返すには、一重引用符を使用してください。たとえば'my string \n my string'、は文字列を返しますmy string \n my string。

- エスケープコードを解析するには二重引用符を使用してください。たとえば"my string \n my new line"、は以下の値を返します。

```
my string
my new line
```

- コメント:#コメントの前に.

  例：

```
Name: MyWorkflow
# This is a comment.
SchemaVersion: 1.0
```

- トリプルダッシュ (---): YAML --- コードでは使用しないでください。 CodeCatalyst 以降はすべて無視されます。---

命名規則

このガイドでは、ワークフロー定義ファイル内の主要項目を指すのにプロパティとセクションという用語を使用します。

- プロパティとは、コロン (:) を含むすべての項目です。たとえば、次のコードスニペットでは、、、、Name、SchemaVersionRunModeTriggersType、のすべてのプロパティがプロパティです。Branches
- セクションとは、サブプロパティを持つすべてのプロパティです。次のコードスニペットには 1 Triggers つのセクションがあります。

> **Note**
>
> このガイドでは、コンテキストによっては「セクション」を「プロパティ」と呼ぶこともあれば、その逆の場合もあります。

```
Name: MyWorkflow
SchemaVersion: 1.0
RunMode: QUEUED
Triggers:
  - Type: PUSH
```

```
    Branches:
      - main
```

トップレベルのプロパティ

以下は、ワークフロー定義ファイル内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していても、別の名前を付けてもかまいません。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に特殊文字やスペースを含めることはできません。

対応する UI: ビジュアルエディター/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)

CodeCatalyst 複数の実行を処理する方法。以下の値のいずれかを使用できます。

- `QUEUED`— 複数の実行がキューに入れられ、次々に実行されます。1 つのキューに実行できるのは最大 50 件です。
- `SUPERSEDED`— 複数の実行がキューに入れられ、次々に実行されます。1 つのキューで実行できるのは 1 回だけなので、2 つの実行が同じキューにまとまると、後の実行が前の実行に優先して (引き継ぎ)、前の実行はキャンセルされます。
- `PARALLEL`— 複数の実行が同時に行われる。

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、「キュー実行、代替実行、parallel 実行の設定」を参照してください。

対応する UI: ビジュアルエディター/ワークフロープロパティ/詳細設定/実行モード

Compute

(オプション)

ワークフローアクションの実行に使用されるコンピューティングエンジン。compute はワークフローレベルまたはアクションレベルのいずれかで指定できますが、両方では指定できません。ワークフローレベルで指定すると、コンピュート設定はワークフローで定義されている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、「アクション間での計算の共有」を参照してください。

コンピュ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環境とランタイム環境の Docker イメージの操作。

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設定されている場合は必須)

コンピュートエンジンのタイプ。以下のいずれかの値を使用できます。

- EC2 (ビジュアルエディター) または EC2 (YAML エディター)

  アクション実行時の柔軟性を考慮して最適化されています。
- Lambda (ビジュアルエディター) または Lambda (YAML エディター)

  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、「コンピュートタイプについて」を参照してください。

対応する UI: ビジュアルエディター/ワークフロープロパティ/アドバンス/コンピュートタイプ

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専用マシンのセットを設定します。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る状態になってい

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プロパティ

を省略した場合、Fleetデフォルトはです。Linux.x86-64.Large

コンピュートフリートの詳細については、を参照してください。コンピュートフリートについて

対応する UI: ビジュアルエディター/ワークフロープロパティ/詳細設定/コンピュートフリート

SharedInstance

(Compute/SharedInstance)

(オプション)

アクションのコンピュート共有機能を指定します。コンピュートシェアリングでは、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) 上で実行されます。以下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、追加の設定がないとアーティファクトや変数などのリソースを共有できないということです。

コンピュートシェアリングの詳細については、「」を参照してくださいアクション間での計算の共有。

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、「トリガーの使用」を参照してください。

対応する UI: ビジュアルエディター/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)

(設定されている場合は必須) Triggers

トリガーの種類を指定します。以下のいずれかの値を使用できます。

- Push (ビジュアルエディター) または PUSH (YAML エディター)

  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るとワークフローの実行を開始します。ワークフロー実行では、プッシュ先のブランチ (つまり、宛先ブランチ) のファイルが使用されます。
- プルリクエスト (ビジュアルエディター) または PULLREQUEST (YAML エディター)

  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開かれたり、更新されたり、閉じられたりしたときに、ワークフローの実行を開始します。ワークフローの実行では、プル元のブランチ (つまり、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ー) または SCHEDULE (YAML エディター)

  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に基づいてワークフローを開始します。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にな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ー) Branches またはプロパティ (YAML エディター) を使用してください)。

  スケジュールトリガーを設定するときは、以下のガイドラインに従ってください。

  - 1 つのワークフローにつき 1 つのスケジュールトリガーのみを使用してください。
  - CodeCatalyst スペースで複数のワークフローを定義している場合、同時に開始するワークフローは 10 個以下にすることをおすすめします。
  - 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ー/ワークフロー図/トリガー/トリガータイプ

Events

(Triggers/Events)

(トリガーがに設定されている場合は必須) Type PULLREQUEST

ワークフローの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た (ビジュアルエディター) または OPEN (YAML エディター)

  ワークフローの実行は、プルリクエストが作成されると開始されます。

- プルリクエストはクローズされます (ビジュアルエディター) または CLOSED (YAML エディター)

  プルリクエストがクローズされると、ワークフローの実行が開始されます。CLOSEDイベントの動作は複雑で、例を見るのが一番わかりやすいでしょう。詳細については、「例: プル、ブランチ、「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ー) または **REVISION** (YAML エディター) に新しいリビジョンが加えられました。

プルリクエストのリビジョンが作成されると、ワークフローの実行が開始されます。プルリクエストが作成されると、最初のリビジョンが作成されます。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。REVISIONプルリクエストのトリガーにイベントを含める場合は、OPENそのイベントを省略できます。というのも、REVISIONはのスーパーセットだからです。OPEN

同じプルリクエストトリガーに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: ビジュアルエディター/ワークフロー図/トリガー/プルリクエスト用イベント

Branches

(Triggers/Branches)

(オプション)

ワークフローの実行をいつ開始すべきかがわかるように、トリガーが監視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。たとえば、main.*で始まるすべてのブランチにマッチさせるにはを使用します。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。

- プッシュトリガーの場合は、プッシュ先のブランチ、つまり宛先ブランチを指定します。マッチしたブランチ内のファイルを使用して、マッチ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
- プルリクエストトリガーには、プッシュ先のブランチ、つまり宛先ブランチを指定します。(マッチしたブランチではなく) 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、マッチ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マッチします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したいファイルを含むブランチを指定します。一致したブランチ内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、一致したブランチごとに 1 つのワークフロー実行が開始されます。

  例: main.*、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監視し、以下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。
>
> - プッシュ先のブランチ (プッシュトリガー用)。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用)。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 つ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ー/ワークフローダイアグラム/トリガー/ブランチ

FilesChanged

(Triggers/FilesChanged)

(トリガーが、またはに設定されている場合はオプション。Type PUSH PULLREQUEST Typeトリガーがに設定されている場合はサポートされませんSCHEDULE。)

ワークフローの実行をいつ開始すべきかを知るために、トリガーが監視するソースリポジトリ内のファイルまたはフォルダーを指定します。正規表現を使用してファイル名やパスを照合できます。

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ー/ワークフロー図/トリガー/ファイルの変更

Expression

(Triggers/Expression)

(トリガーがに設定されている場合は必須) Type SCHEDULE

スケジュールされたワークフローを実行するタイミングを記述する cron 式を指定します。

Cron 式では、次の 6 CodeCatalyst つのフィールド構文を使用します。各フィールドはスペースで区切られます。

```
######## days-of-monthdays-of-week
```

cron 表現の例

分	時間	曜日 (月)	月	曜日	年	意味
0	0	?	*	MON-FRI	*	毎週月曜日から金曜日の深夜 (UTC+0) にワークフローを実行します。
0	2	*	*	?	*	毎日午前 2 時 (UTC+0) にワークフローを実行します。
15	22	*	*	?	*	毎日午後 10 時 15 分 (UTC+0) にワークフローを実行します。
0/30	22-2	?	*	土 - 日	*	土曜日から日曜日の開始日の午後 10 時と翌日の午前 2

分	時間	曜日 (月)	月	曜日	年	意味
						時 (UTC+0) の間、30 分ごとにワークフローを実行します。
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、必ず次のガイドラインに従ってください。 CodeCatalyst

- トリガーごとに SCHEDULE 1 つの cron 式を指定します。
- YAML エディターでは cron 式を二重引用符 (") で囲みます。
- 時刻は協定世界時 (UTC) で指定します。他のタイムゾーンはサポートされていません。
- 実行間隔は 30 分以上に設定してください。より速いケイデンスはサポートされていません。
- *days-of-month*or *days-of-week*フィールドは指定してください。両方は指定しないでください。一方のフィールドに値またはアスタリスク (*) を指定する場合、もう一方のフィールドには疑問符 (?) を使用する必要があります。アスタリスクは「すべて」を意味し、疑問符は「任意」を意味します。

cron 表現のその他の例や?、、などのワイルドカードに関する情報については*、『Amazon EventBridge ユーザーガイド』の「Cron 式リファレンス」を参照してください。L EventBridge との Cron CodeCatalyst 式はまったく同じように機能します。

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ー/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 1 つ以上のアクションのシーケンス。 CodeCatalyst ビルドアクションやテストアクションなど、さまざまな種類の機能を提供する複数のアクションタイプをサポートします。各アクションタイプには以下があります。

- アクション固有のハードコードされた ID `Identifier` を示すプロパティ。たとえば、`aws/build@v1`ビルドアクションを識別します。
- `Configuration`アクション固有のプロパティを含むセクション。

各アクションタイプの詳細については、にあるリファレンスドキュメントを参照してくださいワークフロー定義リファレンス。

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。

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の使用。

```
Name: MyWorkflow
SchemaVersion: 1.0
...
Actions:
  action-name:
    Identifier: action-identifier
    Configuration:
    ...
  #Action groups
  action-group-name:
    Actions:
      ...
```

アクション名

(######) Actions/

(必須)

############################。アクション名はワークフロー内で一意でなければならず、英数字、ハイフン、アンダースコアのみを含む必要があります。構文規則の詳細については、を参照してください。YAML 構文ガイドライン

制限事項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。アクション名

対応する UI: ビジュアルエディタ/ ***######/####/ ##################***

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、異なるアクショングループ間の依存関係を設定することもできます。

*action-group-name*アクショングループに付けたい名前に置き換えてください。アクショングループ名はワークフロー内で一意でなければならず、英数字、ハイフン、アンダースコアのみを使用する必要があります。構文規則の詳細については、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。

対応する UI: なし

ビルドとテストアクションのリファレンス

以下は、ビルドアクションとテストアクションのアクション定義 YAML リファレンスです。2 つのアクションの YAML プロパティは非常に似ているため、1 つのリファレンスがあります。

次のコードで YAML プロパティを選択すると、そのプロパティの説明が表示されます。

> **Note**
>
> 以下の YAML プロパティのほとんどは、ビジュアルエディターに対応する UI 要素を備えています。UI 要素を検索するには、Ctrl+F を使用します。 要素は関連する 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"
```

アクション名

(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なければなりません。アクション名は、英数字 (a-z、A-Z、0-9)、ハイフン (-)、およびアンダースコア (_) 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に特殊文字やスペースを含め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。バージョンを変更する場合以外は、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、「アクションバージョンでの作業」を参照してください。

aws/build@v1ビルドアクションに使用します。

aws/managed-test@v1テストアクションに使用します。

```
#### UI: #######/######/ aws/build @v1 |aws/managed-test @v1 ###
```

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またはアクショングループを指定してください。

「依存する」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アクションの設定

対応する UI:「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。compute はワークフローレベルまたはアクションレベルのいずれかで指定できますが、両方では指定できません。ワークフローレベルで指定すると、コンピュート設定はワークフローで定義されている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、「アクション間での計算の共有」を参照してください。

対応する UI: なし

Type

(**#########**/Compute/)

(Compute含まれている場合は必須)

コンピュートエンジンのタイプ。以下のいずれかの値を使用できます。

- EC2 (ビジュアルエディター) または EC2 (YAML エディター)

  アクション実行時の柔軟性を考慮して最適化されています。
- Lambda (ビジュアルエディター) または Lambda (YAML エディター)

  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、「コンピュートタイプについて」を参照してください。

対応する UI: [設定] タブ/ [コンピュートタイプ]

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専用マシンのセットを設定します。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る状態になっています。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プロパティ

を省略した場合、`Fleet`デフォルトはです。`Linux.x86-64.Large`

対応する UI: [設定] タブ/ [コンピュートフリート]

Timeout

(*######/*) Timeout

(オプション)

CodeCatalyst アクションを終了する前にアクションを実行できる時間を分単位 (YAML エディター) または時間と分 (ビジュアルエディター) で指定します。最小値は 5 分で、最大値はで説明されてい

ます。のワークフローのクォータ CodeCatalyst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)

CodeCatalyst アクションで使用する環境を指定します。

環境の詳細については、「」環境を使用する と「」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。には最大 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AWS アカウント スペースの管理。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)

Amazon S3 や Amazon ECR AWS などのサービスにアクセスして操作するためにこのアクションが使用する IAM ロールの名前を指定します。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てください。アカウント接続に IAM ロールを追加する方法については、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**
>
> ロールに十分な権限があれば、CodeCatalystWorkflowDevelopmentRole-*spaceName*ここでロールの名前を指定できる場合があります。このロールの詳細については、「[アカウントとスペース用の CodeCatalystWorkflowDevelopmentRole-***spaceName*** ロールの作成](#)」を参照してください。CodeCatalystWorkflowDevelopmentRole-*spaceName*ロールには非常に幅広い権限があり、セキュリティ上のリスクが生じる可能性があることを理解してください。このロールは、セキュリティがそれほど問題になら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。

> **Warning**
>
> 権限は、ビルドアクションとテストアクションに必要なものに限定してください。より広範な権限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するキャッシュを指定し、以降のワークフロー実行時にそのキャッシュから復元できるセクション。

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。ファイルキャッシュの操作

対応する UI: [設定] タブ/ [ファイルキャッシュ]-オプション

FileCaching

(######) /Caching/ ***FileCaching***

(オプション)

キャッシュシーケンス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]/[キャッシュを追加]

キー名 1

(#########-1) /Caching/FileCaching/

(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なければなりません。各アクションには最大 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]/[キャッシュの追加]/[キー]

Path

*(**Path**#########-1/) /Caching/FileCaching/*

(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]/[キャッシュの追加]/[パス]

RestoreKeys

*(**RestoreKeys**#########-1/) /Caching/FileCaching/*

(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の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。各キャッシュには最大 5 つのエントリを入れることができますRestoreKeys。

対応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このセクションは、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**
>
> 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) を使用できます。変数はこの合計には含まれません。

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、ソース入力がプライマリ入力で、アーティファクトがセカンダリ入力です。2 次入力のファイルへの参照には、一次入力と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:「入力」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すラベルだけです。

ソースを省略する場合は、その下に少なくとも 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、「ソースの操作」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したい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すでに定義されている必要があります。

入力アーティファクトを何も指定しない場合は、下に少なくとも 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

例を含む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。アーティファクトによる作業

> **Note**
>
> 「アーティファクト-オプション」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ー)、または YAML を検証したときにエラーが発生する場合 (YAML エディター) は、アクションが 1 つの入力しかサポートしていないことが原因である可能性があります。この場合は、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する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 (-)、およびアンダースコア (_) に制限されています。スペースは使用できません。引用符を使用して変数名に特殊文字やスペースを含め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変数の操作。

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るアーティファクトの名前を指定します。Artifact 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にスペース、ハイフン、その他の特殊文字を使用することはできません。

例を含む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。アーティファクトによる作業

対応する UI:「出力」タブ/「アーティファクト」

Name

(######) /Outputs/Artifacts/ ***Name***

(Artifacts - output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。Artifact 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にスペース、ハイフン、その他の特殊文字を使用することはできません。

例を含む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。アーティファクトによる作業

対応する UI:「出力」タブ/「アーティファクト」/「新規出力」/「ビルドアーティファクト名」

Files

*(/Outputs/Artifacts/****Files*****#####)**

(Artifacts - output含まれている場合は必須)

CodeCatalyst アクションによって出力されるアーティファクトに含まれるファイルを指定します。これらのファイルは、ワークフローアクションの実行時に生成され、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配置でき、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。グロブ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

例を含む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による作業。

> **Note**
>
> 検索するアーティファクトまたはソースを示すプレフィックスをファイルパスに追加する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:「出力」タブ/「アーティファクト」/「新規出力」/「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)

アクションでエクスポートする変数を指定して、後続のアクションで使用できるようにします。

例を含む変数の詳細については、を参照してください変数の操作。

対応する UI: [出力] タブ/[変数]/[変数の追加]

変数名 1

(######### 1) /Outputs/Variables/

(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、InputsSteps同じアクションのまたはセクションですで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変数の操作。

対応する UI: [出力] タブ/[変数]/[変数の追加]/[名前]

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、Inputsアクションに渡されたすべてのファイルと、 CodeCatalyst アクション自体によって生成されたすべてのファイルを検索して、テスト、コードカバレッジ、および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はそれぞれ、CodeCatalyst レポートに変換されます。 CodeCatalyst CodeCatalyst CodeCatalyst レポートはサービスに完全に統合されたレポートで、コンソールで表示および操作できます。 CodeCatalyst

> **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。または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。IncludePaths ExcludePaths

対応する UI: [出力] タブ/ [レポート]/[自動検出レポート]

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。

省略した場合、`Enabled`デフォルトはです。`true`

対応する UI: [出力] タブ/ [レポート]/[レポートを自動検出]

ReportNamePrefix

*(/Outputs/AutoDiscoverReports/**ReportNamePrefix**######)*

(が含まれAutoDiscoverReports、有効になっている場合は必須)

CodeCatalyst 見つかったすべてのレポートの先頭に付けるプレフィックスを指定して、 CodeCatalyst 関連するレポートに名前を付けます。たとえば、プレフィックスを指定し`AutoDiscovered`、2 CodeCatalyst つのテストレポートを自動検出すると、`TestSuiteOne.xml``TestSuiteTwo.xml` CodeCatalyst `AutoDiscoveredTestSuiteOne`関連するレポートにはという名前が付けられます。`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

*(##########-1//Outputs/Reports/) **IncludePaths***

(AutoDiscoverReportsが含まれていて有効になっている場合、または含まれている場合は必須)
Reports

CodeCatalyst 未加工レポートを検索する場合に含めるファイルとファイルパスを指定します。たとえば、を指定すると`"/test/report/*"`、 CodeCatalyst `/test/report/*`アクションが使用

するビルドイメージ全体を検索してディレクトリを探します。そのディレクトリが見つかると、CodeCatalyst そのディレクトリでレポートを検索します。

> **Note**
>
> ファイルパスにアスタリスク (*) やその他の特殊文字が 1 つ以上含まれている場合は、そのパスを二重引用符 () 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**
>
> 手動で設定したレポートでは、1 IncludePaths つのファイルに一致するグロブパターン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]/[パスを含める]/[パスを含める]/[パスを含める]
- ***[##] ##/####/########## report-name-1 /######/####/ ######***

ExcludePaths

*(/Outputs/AutoDiscoverReports/**ExcludePaths**######*)

または

*(###########-1//Outputs/Reports/) **ExcludePaths***

(オプション)

CodeCatalyst 未加工レポートを検索するときに除外するファイルとファイルパスを指定します。たとえば、を指定した場合"/test/my-reports/**/*"、 CodeCatalyst /test/my-reports/ディレクトリ内のファイルは検索されません。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**
>
> ファイルパスにアスタリスク (*) やその他の特殊文字が 1 つ以上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]/[レポートを自動検出]/[パスを含める]/[パスを除外]
- ***[##] ##/####/#########/ report-name-1 /###/####/#####***

SuccessCriteria

*(/Outputs/AutoDiscoverReports/**SuccessCriteria**######*)

または

*(##########-1//Outputs/Reports/) **SuccessCriteria***

(オプション)

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)、およびスタティック解析 (SA) レポートの成功基準を指定します。

詳細については、「レポートの達成基準の設定」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

*(/Outputs/Reports/########## 1) /SuccessCriteria/ **PassRate***

(オプション)

CodeCatalyst関連するレポートに合格とマークされるためには、テストレポート内のテストの合格率を指定します。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

*(/Outputs/Reports/########## 1) /SuccessCriteria/ **LineCoverage***

(オプション)

コードカバレッジレポートに含まれる行のうち、対象レポートが合格とマークされるためにカバーする必要がある行の割合を指定します。 CodeCatalyst 有効な値には 10 進数が含まれます。例: `50`、`60.5`。ライン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

*(/Outputs/Reports/########## 1) /SuccessCriteria/ **BranchCoverage***

(オプション)

コードカバレッジレポートに含まれるブランチのうち、 CodeCatalyst 該当するレポートが合格とマークされ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

(/Outputs/Reports/########## 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大度を指定して、関連するレポートに合格とマークしてください。 CodeCatalyst 脆弱性を指定するには、以下を指定する必要があります。

- カウントに含めたい脆弱性の最小重要度。有効な値は、重大度が最も高いものから最も低いものまでCRITICAL、、HIGH、MEDIUMLOW、INFORMATIONALです。

  たとえば、選択した場合はHIGH、HIGHCRITICAL脆弱性が集計されます。
- 指定した重要度で許可したい脆弱性の最大数。この数を超えると、 CodeCatalyst レポートは失敗とマークされます。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・コンポジション分析レポート

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

(/Outputs/Reports/########## 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大度を指定して、 CodeCatalyst 関連するレポートに合格とマークしてください。バグを指定するには、以下を指定する必要があります。

- カウントに含めたいバグの最低重要度。重大度が最も高いものから最も低いものまで、有効な値はCRITICAL、HIGH、MEDIUM、LOW、INFORMATIONALです。

  たとえば、選択した場合HIGH、HIGHCRITICALおよびのバグが集計されます。
- 指定した重大度で許可したいバグの最大数。この数を超えると、 CodeCatalyst レポートは失敗とマークされます。有効な値は整数です。

バグ基準は ESLint SA レポートにのみ適用されます。 PyLint SA レポートの詳細については、を参照してください。スタティック分析レポート

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

*(/Outputs/Reports/########## 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大度を指定して、 CodeCatalyst 関連するレポートに合格とマークしてください。セキュリティ上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。

- カウントに含めたいセキュリティ脆弱性の最小重要度。重大度が最も高いものから最も低いものまで、有効な値はCRITICAL、、HIGH、MEDIUMLOW、INFORMATIONALです。

  たとえば、選択した場合HIGHHIGH、CRITICALセキュリティの脆弱性が集計されます。
- 指定した重大度のセキュリティ脆弱性の許容数の上限。この数を超えると、 CodeCatalyst レポートは失敗とマークされます。有効な値は整数です。

セキュリティ脆弱性基準は ESLint SA PyLint レポートと 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。スタティック分析レポート

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]/[セキュリティ脆弱性]

StaticAnalysisQuality

*(/Outputs/AutoDiscoverReports/SuccessCriteria/**StaticAnalysisQuality**######)*

または

(/Outputs/Reports/########## 1) /SuccessCriteria/ ***StaticAnalysisQuality***

(オプション)

SA レポートで許容される品質問題の最大数と重大度を指定して、 CodeCatalyst 関連するレポートに合格とマークしてください。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。

- カウントに含めたい品質問題の最小重要度。重大度が最も高いものから最も低いものまで、有効な値はCRITICAL、HIGH、MEDIUM、LOW、INFORMATIONALです。

  たとえば、選択した場合はHIGHHIGH、CRITICAL品質に関する問題が集計されます。
- 指定した重要度で許可したい品質問題の最大数。この数を超えると、 CodeCatalyst レポートには不合格とマークされます。有効な値は整数です。

品質問題基準は ESLint SA レポートにのみ適用されます。 PyLint SA レポートの詳細については、を参照してください。スタティック分析レポート

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]/[品質問題]

StaticAnalysisFinding

*(/Outputs/AutoDiscoverReports/SuccessCriteria/***StaticAnalysisFinding***######)*

または

(/Outputs/Reports/########## 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、 CodeCatalyst 関連するレポートに合格とマークしてください。調査結果を指定するには、以下を指定する必要があります。

- カウントに含めたい結果の最小重要度。有効な値は、重大度の高いものから最も低いものまでCRITICAL、、HIGH、MEDIUMLOW、INFORMATIONALです。

  たとえば、選択した場合はHIGH、HIGHCRITICALおよびの結果が集計されます。
- 指定した重要度で許可したい結果の最大数。この数を超えると、 CodeCatalyst レポートには不合格とマークされます。有効な値は整数です。

調査結果は 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スタティック分析レポート。

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]/[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]

レポート名 1

(########### 1) /Outputs/Reports/

(が含まれる場合は必須) Reports

CodeCatalyst 未加工の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]/[レポートの手動設定]/[レポート名]

Format

*(**Format**###########-1/) /Outputs/Reports/*

(が含まれる場合は必須) Reports

レポートに使用しているファイル形式を指定してください。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ー) または CUCUMBERJSON (YAML エディター) を指定します。
  - JUnit XML には JUnit (ビジュアルエディター) または JUNITXML (YAML エディター) を指定します。

  - NUnit XML には NUnit (ビジュアルエディター) または NUNITXML (YAML エディター) を指定します。
  - NUnit 3 XML の場合は NUnit3 (ビジュアルエディター) または (YAML エディター) を指定します。NUNIT3XML
  - Visual Studio TRX の場合は、Visual Studio TRX (ビジュアルエディター) または (YAML エディター) を指定します。VISUALSTUDIOTRX
  - TestNG XML の場合は、TestNG (ビジュアルエディター) または TESTNGXML (YAML エディター) を指定します。
- コードカバレッジレポートの場合:
  - Clover XML の場合は、Clover (ビジュアルエディター) または CLOVERXML (YAML エディター) を指定します。
  - Cobertura XML の場合は、Cobertura (ビジュアルエディター) または (YAML エディター) を指定します。COBERTURAXML
  - JaCoCo XML の場合は JaCoCo(ビジュアルエディター) または (YAML エディター) を指定します。JACOCOXML
  - SimpleCov simplecov-json ではなく simplecov によって生成された JSON には、Simplecov (ビジュアルエディター) または (YAML エディター) を指定します。SIMPLECOV
- 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ー) または SARIFSCA (YAML エディター) を指定します。

UI: [##] ##/####/####/####/#######/##/ report-name-1/[#######] # [#######]

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。付属のアクティブなイメージのいずれかを指定することも CodeCatalyst、独自のイメージを使用することもできます。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配置できます。Docker イメージを指定しない場合、アクションはアクティブなイメージの 1 つを処理に使用します。どのアクティブイメージがデフォルトで使用されるかについては、を参照してくださいアクティブイメージ。

独自の Docker イメージを指定する方法の詳細については、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ーを指定します。有効な値を次に示します。

- CODECATALYST(YAML エディター)

  CodeCatalyst イメージはレジストリに保存されます。
- Docker Hub (ビジュアルエディター) または DockerHub (YAML エディター)

  イメージは Docker Hub イメージレジストリーに保存されます。
- その他のレジストリー (ビジュアルエディター) または Other (YAML エディター)

  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ーならどれでも使用できます。
- Amazon エラスティックコンテナレジストリ (ビジュアルエディタ) または ECR (YAML エディタ)

  イメージは Amazon エラスティックコンテナレジストリのイメージリポジトリに保存されます。Amazon ECR リポジトリ内のイメージを使用するには、このアクションが 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、以下の権限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。(必要に応じて、既存のロールを変更してアクセス権限とポリシーを含めることができます)。

  IAM ロールのロールポリシーには、以下のアクセス権限が含まれている必要があります。

  - ecr:BatchCheckLayerAvailability

- ecr:BatchGetImage
- ecr:GetAuthorizationToken
- ecr:GetDownloadUrlForLayer

IAM ロールには以下のカスタム信頼ポリシーが含まれている必要があります。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service":  [
                   "codecatalyst-runner.amazonaws.com",
                   "codecatalyst.amazonaws.com"
                ]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

IAM ロールの作成に関する詳細については、『IAM ユーザーガイド』の「カスタム信頼ポリシーを使用してロールを作成する (コンソール)」を参照してください。

ロールを作成したら、環境を通じ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 エラスティックコンテナレジストリ、Docker Hub、およびその他のレジストリオプション

Image

(*/Configuration/Container/**Image**######*)

(Container含まれている場合は必須)

次のいずれかを指定します。

- CODECATALYSTレジストリーを使用している場合は、イメージを以下のアクティブ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 レジストリー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、カスタムレジストリが期待する値にイメージを設定します。

対応する UI: ランタイム環境の docker イメージ (レジストリが Docker Hub 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 レジストリの場合)、およびイメージ 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 プロジェクトのビルド方法の例を以下に示します。

```
Steps:
  - Run: npm install
  - Run: npm run build
```

ファイルパスを指定する方法の例を以下に示します。

```
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について詳しくは、「」ソースリポジトリ内のファイルを参照する と「」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を解決するため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です。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して共有できます。

パッケージの詳細については、を参照してください内のパッケージ CodeCatalyst。

対応する UI: [設定] タブ/ [パッケージ]

NpmConfiguration

(*/Configuration/Packages/**NpmConfiguration**######*)

(Packages含まれている場合は必須)

npm パッケージフォーマットの設定を定義するセクション。この設定は、ワークフローの実行中に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含まれている場合は必須)

CodeCatalyst アクションに使用させたい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。Package リポジトリ

対応UI: 設定タブ/Package /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したい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と、指定したパッケージリポジトリが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。そのスコープのパッケージが npm クライアントからリクエストされた場合、npm クライアントはデフォルトの代わりにそのリポジトリを使用します。各スコープ名の先頭には「@」を付ける必要があります。

オーバーライドするスコープを含めると、最後のリポジトリが優先されます。

省略すると、Scopes指定し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。

スコープについて詳しくは、「スコープ付きパッケージ」を参照してくださいPackage 名前空間。

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

「Amazon S3 公開」アクションリファレンス

Amazon S3 公開アクションのアクション定義 YAML リファレンスを次に示します。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「Amazon S3 パブリッシュ」アクションの追加。

> **Note**
>
> 次の YAML プロパティのほとんどには、ビジュアルエディタに対応する UI 要素があります。UI 要素を検索するには、Ctrl+F を使用します。 要素は、関連する 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と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、「アクションバージョンでの作業」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 diagram/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には、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。

「依存関係」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アクションの設定。

対応する UI: の入力タブ/依存値 - オプション

Compute

(*S3Publish*/Compute)

(オプション)

ワークフローアクションの実行に使用されるコンピューティングエンジン。ワークフローレベルまたはアクションレベルでコンピューティングを指定できますが、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、ワークフローで定義されたすべてのアクションにコンピューティング設定が適用されます。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、「アクション間での計算の共有」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、「コンピュートタイプについて」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専用マシンのセットを設定します。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 (YAML 工ディタ) 、または時間と分単位 (ビジュアル工ディタ) で指定します。最小値は 5 分で、最大値は で説明されていますのワークフローのクォータ CodeCatalyst。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) に存在す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で、アーティファクトはセカンダリ入力です。セカンダリ入力のファイルへの参照には、プライマリから配布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: 入力タブ

Sources

(*S3Publish*/Inputs/Sources)

(Amazon S3 に公開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須)

Amazon S3 に公開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 に発行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、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、「ソースの操作」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 に公開するファイルが、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須)

Amazon S3 に発行するファイルが、以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による作業。

対応する UI: 設定タブ/Artifacts - 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する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。引用符を使用して、変数名で特殊文字とスペースを有効に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変数の操作。

対応する UI: 入力タブ/可変 - 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、環境を使用する「」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。では、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AWS アカウント スペースの管理。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 にアクセスしてファイル AWS をコピーするために使用する 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されているものに制限します。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**
>
> CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールの名前は、必要に応じてここで指定できます。このロールの詳細については、「アカウントとスペース用の CodeCatalystWorkflowDevelopmentRole-***spaceName*** ロールの作成」を参照してください。CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールには、セキュリティ上の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、セキュリティ上の問題が少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

(*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以前のアクションのアーティファクトに存在でき、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、 の内容が `/myFolder` 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 にのみ`myfile.txt`コピーを`./myFolder/myfile.txt`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、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/ソースパス

DestinationBucketName

(*S3Publish*/Configuration/DestinationBucketName)

(必須)

ファイルを公開する 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)

ファイルを公開する 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

AWS CDK 「ブートストラップ」アクションリファレンス

以下は、AWS CDKブートストラップアクションのアクション定義 YAML リファレンスです。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「AWS CDK ブートストラップ」アクションの追加。

> **Note**
>
> 次の YAML プロパティのほとんどには、ビジュアルエディタに対応する UI 要素があります。UI 要素を検索するには、Ctrl+F を使用します。 要素は、関連する 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と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、「アクションバージョンでの作業」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 diagram/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には、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。

「依存関係」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アクションの設定。

対応する UI: での入力タブ/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。ワークフローレベルまたはアクションレベルでコンピューティングを指定できますが、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、ワークフローで定義されたすべてのアクションにコンピューティング設定が適用されます。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、「アクション間での計算の共有」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、「コンピュートタイプについて」を参照してください。

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専用マシンのセットを設定します。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 (YAML エディタ) 、または時間と分単位 (ビジュアルエディタ) で指定します。最小値は 5 分で、最大値は で説明されていますのワークフローのクォータ CodeCatalyst。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: 入力タブ

> **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 アプリケーション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、「ソースの操作」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須 ）

AWS CDK アプリが以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。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による作業。

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、他のアクションの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による作業。

対応する UI: タブ/アーティファクトを出力する

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。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、テストまたはトラブルシューティングの目的でレコードを保持することを検討してください。

対応する UI: タブ/Artifacts/Add artifact/Build artifact name を出力します。

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: ビルドによって生成されたタブ/Artifacts/Add artifact/Files を出力します。

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、環境を使用する「」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/'Environment/account/role'/Environment

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/'Environment/account/role'/Environment

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。では、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AWS アカウント スペースの管理。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表示されるアクセス許可は、 `cdk bootstrap` コマンドがブートストラップを実行するために必要なものです。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可能性があります。

> **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。これは非常に許容度が高く、他のアクションと一緒に使用すると、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されているものに制限します。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、利用可能になった後にリソース名を使用して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**
>
> CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールの名前は、必要に応じてここで指定できます。このロールの詳細については、「アカウントとスペース用の CodeCatalystWorkflowDevelopmentRole-***spaceName*** ロールの作成」を参照してください。CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールには、セキュリティ上の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、セキュリティ上の問題が少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/'Environment/account/role'/Role

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するAWS リージョン先の を指定します。このリージョンは、AWS CDKアプリケーションのデプロイ先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) (AWS CDKツールキットとも呼ばれます) の完全バージョン。例えば、`2.102.1` などです。アプリケーションの構築およびデプロイ時に一貫性と安定性を確保するため、完全な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 の最新機能と修正を利用する`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 CLI (または最新バージョン) AWS CDK 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AWS環境のブートストラップ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。

- 「AWS CDK デプロイ」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 CDKCLI バージョン

「AWS CDK デプロイ」アクションリファレンス

以下は、AWS CDK デプロイアクションのアクション定義 YAML リファレンスです。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「AWS CDK デプロイ」アクションの追加。

> **Note**
>
> 以下の YAML プロパティのほとんどは、ビジュアルエディターに対応する UI 要素を備えています。UI 要素を検索するには、Ctrl+F を使用します。 要素は関連する YAML プロパティとともに一覧表示されます。

```yaml
# 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/cdk.json
      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なければなりません。アクション名は、英数字 (a-z、A-Z、0-9)、ハイフン (-)、およびアンダースコア (_) 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に特殊文字やスペースを含め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。バージョンを変更する場合以外は、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、「アクションバージョンでの作業」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)

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。AWS CDK 以下のように、AWS CDK **DependsOn**ブートストラップアクションをプロパティに指定することをおすすめします。

```
CDKDeploy:
  Identifier: aws/cdk-deploy@v1
  DependsOn:
    - CDKBootstrap
```

> **Note**
>
> ブートストラップはアプリをデプロイするための必須の前提条件です。 AWS CDK AWS CDK ブートストラップアクションをワークフローに含めない場合は、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デプロイする別の方法を見つける必要があります。AWS CDK 詳細については、「「AWS CDK デプロイ」アクションの追加」の 前提条件 を参照してください。

「depends on」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アクションの設定

対応する UI:「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の実行に使用されるコンピューティングエンジン。compute はワークフローレベルまたはアクションレベルのいずれかで指定できますが、両方では指定できません。ワークフローレベルで指定すると、コンピュート設定はワークフローで定義されている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、「アクション間での計算の共有」を参照してください。

対応する UI: なし

Type

(*CDKDeploy*/Compute/Type)

(Compute含まれている場合は必須)

コンピュートエンジンのタイプ。以下のいずれかの値を使用できます。

- EC2 (ビジュアルエディター) または EC2 (YAML エディター)

  アクション実行時の柔軟性を考慮して最適化されています。
- Lambda (ビジュアルエディター) または Lambda (YAML エディター)

  アクションの起動速度が最適化されました。

コンピューティングタイプの詳細については、「コンピュートタイプについて」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]/[コンピュートタイプ]

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専用マシンのセットを設定します。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る状態になっています。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プロパティ

を省略した場合、`Fleet`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]/[コンピュートフリート]

Timeout

(*CDKDeploy*/Timeout)

(必須)

アクションが終了するまでにアクションを実行できる時間を分単位 (YAML エディター) または時間と分 (ビジュアルエディター) で指定します。 CodeCatalyst 最小値は 5 分で、最大値はで説明されています。のワークフローのクォータ CodeCatalyst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`このセクションは、`CDKDeploy`ワークフローの実行中に必要なデータを定義します。

> **Note**
>
> AWS CDK 各デプロイアクションで使用できる入力は 1 つだけです (ソースまたはアーティファクト)。

対応する UI:「入力」タブ

Sources

(*`CDKDeploy`*/Inputs/Sources)

(AWS CDK デプロイする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統合します。現在、`WorkflowSource`サポートされているラベルはだけです。

AWS CDK アプリケーション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含まれている必要があります。

sources の詳細については、「ソースの操作」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)

(AWS CDK デプロイする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は必須)

AWS CDK 前のアクションで生成されたアーティファクトにアプリが含まれている場合は、そのアーティファクトをここで指定します。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、 CloudFormation指定されたアーティファクト内のアプリをテンプレートに統合します。 AWS CDK アプリケーション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。

例を含む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。アーティファクトによる作業

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る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、他のアクションの入力として参照できます。

例を含む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。アーティファクトによる作業

対応する UI:「出力」タブ/「アーティファクト」

Name

(*CDKDeploy*/Outputs/Artifacts/Name)

(Artifacts - output含まれている場合は必須)

AWS CloudFormation AWS CDK 実行時にデプロイアクションによって合成される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しません。テストやトラブルシューティングの目的で、合成したテンプレートをアーティファクトに保存して、その記録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」/「アーティファクトを追加」/「アーティファクト名をビルド」

Files

(*CDKDeploy*/Outputs/Artifacts/Files)

(Artifacts - output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。"cdk.out/**/*" AWS CDK AWS CloudFormation アプリの合成テンプレートを含めるよう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ファイル以外の出力ディレクトリを指定した場合は、cdk.outcdk.outの代わりにそのディレクトリをここに指定します。

対応する UI: [出力] タブ/ [アーティファクト]/[アーティファクトの追加]/[ビルドによって生成されたファイル]

Environment

(*CDKDeploy*/Environment)

(必須)

アクションで使用する環境を指定します。 CodeCatalyst

環境の詳細については、「」環境を使用する と「」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。には最大 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AWS アカウント スペースの管理。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]/[アカウント接続AWS]

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]/[アカウント接続AWS]

Role

(*CDKDeploy*/Environment/Connections/Role)

(必須)

デプロイアクションがアプリケーションスタックへのアクセスとデプロイに使用する IAM ロールの名前を指定します。AWS CDK AWS AWS CDK このロールには以下が含まれていることを確認してください。

- 以下のアクセス権限ポリシー:

> **Warning**
> 権限を次のポリシーに示されているものに制限します。より広範な権限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-*"
        }
    ]
}
```

- 以下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てください。アカウント接続に IAM ロールを追加する方法の詳細については、「」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、CodeCatalystWorkflowDevelopmentRole-*spaceName*ここで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用の CodeCatalystWorkflowDevelopmentRole-***spaceName*** ロールの作成」を参照してください。CodeCatalystWorkflowDevelopmentRole-*spaceName*ロールには非常に幅広い権限があり、セキュリティ上のリスクが生じる可能性があることを理解してください。このロールは、セキュリティがそれほど問題になら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]

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 AWS CDK アプリ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アプリスタックの名前。bin***##### TypeScript##***エントリポイントファイルが別の言語で書かれている場合は、似たような内容になります。

```
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**
>
> スタックが複数ある場合は、スタックをネストした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 リージョン AWS CDK アプリケーションスタックをデプロイする先を指定します。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 AWS CDK アプリケーションスタック内の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だけでなく、スタック内の個々のリソースにも適用されます。タグ付けについて詳しくは、『AWS Cloud Development Kit (AWS CDK) 開発者ガイド』の「タグ付け」を参照してください。

対応する UI: [設定] タブ/詳細設定-オプション/ [タグ]

Context

(*CDKDeploy*/Configuration/Context)

(オプション)

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。 AWS CDK コンテキストについて詳しくは、『開発者ガイド』の「ランタイムコンテキスト」を参照してください。AWS Cloud Development Kit (AWS CDK)

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。AWS CDK

次のいずれかを指定します。

- AWS Cloud Development Kit (AWS CDK) このアクションで使用したいコマンドラインインターフェイス (CLI) (AWS CDK ツールキットとも呼ばれる) のフルバージョン。例えば、`2.102.1` などです。アプリケーションをビルドしてデプロイするときの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 `latest` の最新の機能や修正点を活用するように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 (または最新バージョン) の AWS CDK CLI CodeCatalyst を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に必要なコマンドを実行します。 AWS

使用できる CDK CLI のサポート対象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、アクションは次のトピックのいずれかに説明されているデフォルト AWS CDK CLI バージョンを使用します。

- 「AWS CDK デプロイ」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 プロジェクトのファイルを含むディレクトリへのパス。cdk.jsonAWS CDK デプロイアクションはこのフォルダーから実行され、アクションによって作成された出力はすべてこのディレクトリに追加されます。指定しない場合、AWS CDK **cdk.json**デプロイアクションはファイルがプロジェクトのルートにあると仮定します AWS CDK 。

対応する UI:「設定」タブ/ cdk.json が置かれているディレクトリ

CfnOutputVariables

(*CDKDeploy*/Configuration/CfnOutputVariables)

(オプション)

CfnOutput AWS CDK アプリケーションコード内のどの構成をワークフロー出力変数として公開するかを指定します。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変数の操作。

たとえば、 AWS CDK アプリケーションコードが次のようになっているとします。

```
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',
    });
    ...
  }
}
```

... すると、CfnOutputVariablesプロパティは次のようになります。

```
Configuration:
  ...
  CfnOutputVariables: '["bucketName","tableName"]'
```

... すると、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
tableName	table.tableName

その後、bucketNametableName以降のアクションでと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定義済みの変数を参照する。。

CfnOutputCfnOutputVariablesプロパティに構成を何も指定しない場合、アクションは見つかった最初の 4 つ (またはそれ以下) CloudFormation の出力変数をワークフロー出力変数として公開します。詳細については、「AWS CDKアクション変数を「デプロイ」します。」を参照してください。

> **Tip**
>
> CloudFormation アクションが生成するすべての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] タブを調べます。ログには、 CloudFormation AWS CDK アプリに関連するすべての出力変数のリストが含まれています。 CloudFormation すべての変数がわかったら、CfnOutputVariablesプロパティを使用してどの変数をワークフロー出力変数に変換するかを指定できます。

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 class CfnOutput (CfnOutputconstruct) にあるコンストラクトのドキュメントを参照してください。

対応する UI: [設定] AWS CloudFormation タブ/出力変数

CloudAssemblyRootPath

(*CDKDeploy*/Configuration/CloudAssemblyRootPath)

(オプション)

(cdk synthオペレーションを使用して) AWS CDK アプリのスタックを既にクラウドアセンブリ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 (cdk.out) のルートパスを指定します。AWS CloudFormation 指定したクラウドアセンブリディレクトリにあるテンプレートは、AWS CDK deploy AWS アカウント cdk deploy --app アクションによってコマンドを使用してユーザーにデプロイされます。--appオプションが存在する場合、cdk synth操作は行われません。

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 **cdk deploy--app**デプロイアクションはオプションなしで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 AWS CDK オペレーションはアプリを合成 (cdk synth) してにデプロイします AWS アカウント。

「AWS CDK deploy」アクションが実行時に合成を実行できるのに、なぜ既存の合成済みクラウドアセンブリを指定する必要があるのでしょうか。

合成された既存のクラウド・アセンブリを以下のように指定したい場合があります。

- 「AWS CDK 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デプロイされるようにしてください。

  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が実行されるタイミングに応じて異なるファイルを合成してデプロイする可能性があります。たとえば、AWS CDK デプロイアクションは、テスト段階ではクラウドアセンブリを 1 つの依存関係セットと合成し、運用段階では別の依存関係セットを持つクラウドアセンブリを合成することがあります (ステージ間で依存関係が変更された場合)。テスト対象とデプロイされる内容が完全に一致することを保証するには、一度合成してから、Path to cloud assembly directory Directory フィールド (ビジュアルエディター) CloudAssemblyRootPath またはプロパティ (YAML エディター) を使用して、すで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。
- アプリでは非標準のパッケージマネージャーやツールを使用してください。 AWS CDK

synth操作中、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。これらのツールを使用してアクションが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、 AWS CDK `cdk.json`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アプリのファイルに指定し、AWS CDK デプロイアクションを含まない方法で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ディレクトリへのパス」フィールド (ビジュアルエディター) `CloudAssemblyRootPath` またはプロパティ (YAML エディター) で指定できます。

`cdk.json`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むようにファイルを設定する方法については、「 AWS CDK アプリコマンドの指定」を参照してください。

`cdk deploy`、`cdk synth`およびコマンド、*`--app`*およびオプションについては、『開発者ガイド』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。AWS Cloud Development Kit (AWS CDK)

クラウドアセンブリについて詳しくは、API リファレンスの「Cloud Assembly」を参照してください。AWS Cloud Development Kit (AWS CDK)

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]

AWS Lambda 「呼び出し」アクションリファレンス

以下は、AWS Lambda 呼び出しアクションのアクション定義 YAML リファレンスです。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AWS Lambda 「呼び出し」アクションの追加。

> **Note**
>
> 次の YAML プロパティのほとんどには、ビジュアルエディタに対応する UI 要素があります。UI 要素を検索するには、Ctrl+F を使用します。 要素は、関連する 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と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、「アクションバージョンでの作業」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 diagram/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には、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。

「依存関係」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アクションの設定。

対応する UI: の入力タブ/依存値 - オプション

Compute

(*LambdaInvoke*/Compute)

(オプション)

ワークフローアクションの実行に使用されるコンピューティングエンジン。ワークフローレベルまたはアクションレベルでコンピューティングを指定できますが、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、ワークフローで定義されたすべてのアクションにコンピューティング設定が適用されます。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、「アクション間での計算の共有」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、「コンピュートタイプについて」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専用マシンのセットを設定します。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)

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 (YAML 工ディタ) 、または時間と分単位 (ビジュアル工ディタ) で指定します。最小値は 5 分で、最大値は で説明されていますのワークフローのクォータ CodeCatalyst。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が必要とする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: 入力タブ

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、「ソースの操作」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による作業。

対応する UI: 設定タブ/Artifacts - 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する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。引用符を使用して、変数名で特殊文字とスペースを有効に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変数の操作。

対応する UI: 入力タブ/可変 - 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、環境を使用する「」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/'Environment/account/role'/Environment

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。では、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AWS アカウント スペースの管理。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 関数にアクセスして AWS 呼び出すために使用する 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されているものに制限します。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**
>
> CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールの名前は、必要に応じてここで指定できます。このロールの詳細については、「アカウントとスペース用の

CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールの作成」を参照してください。CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールには、セキュリティ上の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、セキュリティ上の問題が少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「Invoke」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格納できます。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からの相対パスです。

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 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`true`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/エラー発生時の継続

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 Tail 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「Invoke」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、「ワークフロー実行の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変数の操作。

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 変数nameと 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 レスポンスの各最上位キーを出力変数に変換します。詳細については、「「AWS Lambdainvoke」アクション変数」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出力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/応答フィルター - 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、他のアクションの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による作業。

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します。

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、アク

ションのログに Lambda レスポンスペイロードを表示できます。このログは、 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、「ワークフロー実行の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: 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します。

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/Artifacts/ファイルを出力します。

AWS CloudFormation 「スタックのデプロイ」アクションリファレンス

AWS CloudFormation デプロイスタックアクションのアクション定義 YAML リファレンスを次に示します。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「 AWS CloudFormation デプロイスタック」アクションの追加。

> **Note**
>
> 次の YAML プロパティのほとんどには、ビジュアルエディタに対応する UI 要素があります。UI 要素を検索するには、Ctrl+F を使用します。 要素は、関連する 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と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、「アクションバージョンでの作業」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には、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。

「依存関係」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アクションの設定。

対応する UI: の入力タブ/依存値 - 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。ワークフローレベルまたはアクションレベルでコンピューティングを指定できますが、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、ワークフローで定義されたすべてのアクションにコンピューティング設定が適用されます。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、「アクション間での計算の共有」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、「コンピュートタイプについて」を参照してください。

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専用マシンのセットを設定します。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 を終了する前にアクションを実行できる時間を分単位 (YAML エディタ) 、または時間と分単位 (ビジュアルエディタ) で指定します。最小値は 5 分で、最大値は で説明されていますのワークフローのクォータ CodeCatalyst。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、環境を使用する「」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/'Environment/account/role'/Environment

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/'Environment/account/role'/Environment

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。では、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AWS アカウント スペースの管理。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)

デプロイAWS CloudFormationスタックアクションが AWSおよびAWS CloudFormation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。

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されているものに制限します。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、利用可能になった後にリソース名を使用して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**
>
> CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールの名前は、必要に応じてここで指定できます。このロールの詳細については、「アカウントとスペース用の CodeCatalystWorkflowDevelopmentRole-***spaceName*** ロールの作成」を参照してください。CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールには、セキュリティ上の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、セキュリティ上の問題が少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/'Environment/account/role'/Role

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) に存在す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で、アーティファクトはセカンダリ入力です。セカンダリ入力のファイルへの参照には、プライマリから配布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: 入力タブ

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は必須)

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、「ソースの操作」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須)

デプロイする CloudFormation または AWS SAM テンプレートが、以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による作業。

対応する UI: 設定タブ/Artifacts - 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)

Deploy CloudFormation スタック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。AWS CloudFormation

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 リージョン先の 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. CloudFormation uses の Amazon リソースネーム (ARN) を指定して、スタック内のリソースにアクセスして変更します。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。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「」で説明されているチュートリアルに従った場合、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を一覧表示スタックロールを作成するにはする「」というタイトルの手順が含まれています。

  > **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、YAML エディタを使用して `role-arn`フィールドでロール ARN を指定できます。

> **Note**
>
> CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールの名前は、必要に応じてここで指定できます。このロールの詳細については、「アカウントとスペース用の CodeCatalystWorkflowDevelopmentRole-***spaceName*** ロールの作成」を参照してください。CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールには、セキュリティ上の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、セキュリティ上の問題が少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AWS CloudFormationを作成できるようにするために必要な 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。

デプロイAWS CloudFormationスタックアクションのログ##[error] requires capabilities: [*capability-name*]に が表示された場合は、IAM 機能のエラーを修正する方法を教えてください。「」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)

デフォルト値がない、またはデフォルト値以外の値を指定する AWS CloudFormationまたは AWS SAM テンプレートでパラメータを指定します。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」を参照してください。 AWS CloudFormation

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てください。

```
{
  "Parameters": {
    "Param1": "Value1",
    "Param2": "Value2",
    ...
  }
}
```

または...

```
[
  {
     "ParameterKey": "Param1",
     "ParameterValue": "Value1"
  },
  ...
]
```

（他の構文もありますが、記述 CodeCatalyst 時に 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、AWS CLI「コマンドリファレンス」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存在する場合は、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名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テンプレートがあるとします。

```
##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**
>
> 値undefinedとして を使用すると、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`
> 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 を使用することになります。

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/アドバンスト/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/アドバンスト/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。変更セットが適切であると判断した場合 CodeCatalyst 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、AWS CLI「 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。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/アドバンスト/実行変更セットなし

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 CodeCatalyst 、デプロイAWS CloudFormationスタックアクションを失敗させるかどうかを指定します。(変更セットが空の場合は、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は 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、AWS CLI「 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。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/アドバンスト/空の変更セットで失敗する

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、スタックを最後の既知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、AWS CLI「 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 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、「 AWS CloudFormationユーザーガイド」の「スタック障害オプション」を参照してください。

対応する UI: 設定タブ/アドバンスト/無効化ロールバック

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/アドバンスト/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、スタックのステータスを に設定するまでに が割り当て CloudFormation る時間を分単位で指定しますCREATE_FAILED。 CloudFormation が割り当てられた時間内にスタック全体を作成できない場合、スタックの作成がタイムアウトで失敗し、スタックはロールバックされます。

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/アドバンスト/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`」と入力します。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/アドバンスト/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`」と入力します。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ための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/モニタリングアラーム 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 モニタリングする時間の長さを 0～180 分の間で指定します。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が CloudFormation ロールバックされます。

デフォルト: 0. は、スタックリソースのデプロイ中にアラーム CloudFormation をモニタリングし、その後は監視しません。

対応する UI: 設定タブ/アドバンスト/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、「Linux インスタンス用 Amazon EC2 ユーザーガイド」の「 リソースのタグ付けAmazon EC2」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/アドバンスト/タグ

「Amazon ECS へのデプロイ」アクションリファレンス

以下は、Amazon ECS へのデプロイアクションのアクション定義 YAML リファレンスです。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「Amazon ECS にデプロイ」アクションの追加。

> **Note**
>
> 次の YAML プロパティのほとんどには、ビジュアルエディタに対応する UI 要素があります。UI 要素を検索するには、Ctrl+F を使用します。 要素は、関連する 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と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、「アクションバージョンでの作業」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には、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。

「依存関係」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アクションの設定。

対応する UI: での入力タブ/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。ワークフローレベルまたはアクションレベルでコンピューティングを指定できますが、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、ワークフローで定義されたすべてのアクションにコンピューティング設定が適用されます。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、「アクション間での計算の共有」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、「コンピュートタイプについて」を参照してください。

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専用マシンのセットを設定します。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

(*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 (YAML エディタ) 、または時間と分単位 (ビジュアルエディタ) で指定します。最小値は 5 分で、最大値は で説明されていますのワークフローのクォータ CodeCatalyst。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、環境を使用する「」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/'Environment/account/role'/Environment

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/'Environment/account/role'/Environment

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。では、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AWS アカウント スペースの管理。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されているものに制限します。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、利用可能になった後にリソース名を使用して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**
>
> CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールの名前は、必要に応じてここで指定できます。このロールの詳細については、「アカウントとスペース用の

CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールの作成」を参照してください。CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールには、セキュリティ上の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、セキュリティ上の問題が少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

(*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 がECSDeployAction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: 入力タブ

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は必須)

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、「ソースの操作」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須）

デプロイするタスク定義ファイルが、以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による作業。

対応する UI: 設定タブ/Artifacts - 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。Deploy to Amazon ECS 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」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存在する場合、パスはソースリポジトリのルートフォルダからの相対パスです。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の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」を参照してください。 AWS CodeDeploy

> **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 情報を指定します。ローリング更新デプロイ (デフォルト) の場合は、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 ビジュアルエディタではフィールドが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 CodeDeploy ビジュアルエディターに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、「 ユーザーガイド」の「 でのデプロイの使用 CodeDeploy」を参照してください。 AWS CodeDeploy

対応する UI: 設定タブ/CodeDeploy デプロイの説明

「Kubernetes クラスターへのデプロイ」アクションリファレンス

Kubernetes クラスターへのデプロイアクションのアクション定義 YAML リファレンスを次に示します。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「Kubernetes クラスターへのデプロイ」アクションの追加。

> **Note**
>
> 次の YAML プロパティのほとんどには、ビジュアルエディタに対応する UI 要素があります。UI 要素を検索するには、Ctrl+F を使用します。 要素は、関連する 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と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、「アクションバージョンでの作業」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 diagram/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には、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。

「依存関係」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アクションの設定。

対応する UI: の入力タブ/依存値 - オプション

Compute

(*DeployToKubernetesCluster*/Compute)

(オプション)

ワークフローアクションの実行に使用されるコンピューティングエンジン。ワークフローレベルまたはアクションレベルでコンピューティングを指定できますが、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、ワークフローで定義されたすべてのアクションにコンピューティング設定が適用されます。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、「アクション間での計算の共有」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、「コンピュートタイプについて」を参照してください。

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専用マシンのセットを設定します。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

(*DeployToKubernetesCluster*/Timeout)

(オプション)

がアクション CodeCatalyst を終了する前にアクションを実行できる時間を分単位 (YAML エディタ)、または時間と分単位 (ビジュアルエディタ) で指定します。最小値は 5 分で、最大値は「」で説明されていますのワークフローのクォータ CodeCatalyst。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、環境を使用する「」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/'Environment/account/role'/Environment

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/'Environment/account/role'/Environment

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。では、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AWS アカウント スペースの管理。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 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されているものに制限します。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、利用可能になった後にリソース名を使用して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。

- アカウントの接続。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**
>
> アカウント接続と に IAM ロールを追加する手順チュートリアル:Amazon EKS へのアプリケーションのデプロイについては、「」も参照してください ConfigMap。

> **Note**
>
> CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールの名前は、必要に応じてここで指定できます。このロールの詳細については、「アカウントとスペース用の CodeCatalystWorkflowDevelopmentRole-***spaceName*** ロールの作成」を参照してください。CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールには、セキュリティ上の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、セキュリティ上の問題が少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、必ずロールを ConfigMap 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: 入力タブ

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は必須）

Kubernetes 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、「ソースの操作」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須）

Kubernetes マニフェストファイル、または以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による作業。

対応する UI: 設定タブ/Artifacts - 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 アプリケーションのデプロイ先となる 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 (1 つ) へのパスを指定します。これは、Kubernetes ドキュメントの「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、ファイル名の先頭に数字または文字を増やして処理順序を制御するようにしてください。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存在する場合、パスはソースリポジトリのルートフォルダからの相対パスです。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の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

GitHub 「アクション」アクションリファレンス

以下は、Actions アクションのアクション定義 YAML GitHub リファレンスです。

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**
>
> 次の YAML プロパティのほとんどには、ビジュアルエディタに対応する UI 要素があります。UI 要素を検索するには、Ctrl+F を使用します。 要素は、関連する YAML プロパティとともに一覧表示されます。

```yaml
# 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と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、「アクションバージョンでの作業」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には、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。

「依存関係」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アクションの設定。

対応する UI: の入力タブ/依存値 - 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。ワークフローレベルまたはアクションレベルでコンピューティングを指定できますが、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、ワークフローで定義されたすべてのアクションにコンピューティング設定が適用されます。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、「アクション間での計算の共有」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専用マシンのセットを設定します。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)

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 (YAML エディタ)、または時間と分単位 (ビジュアルエディタ) で指定します。最小値は 5 分で、最大値は で説明されていますのワークフローのクォータ CodeCatalyst。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、環境を使用する「」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/'Environment/account/role'/Environment

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。では、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AWS アカウント スペースの管理。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)

Amazon S3 や Amazon ECR などの AWSのサービスにアクセスして操作するために、このアクションが使用する 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。IAM ロールをアカウント接続に追加するには、「」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る場合があります。このロールの詳細については、「アカウントとスペース用のCodeCatalystWorkflowDevelopmentRole-***spaceName*** ロールの作成」を参照してください。CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールには、セキュリティ上の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、セキュリティ上の問題が少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**
>
> アクセス許可を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**
>
> アクション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) に存在す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で、アーティファクトはセカンダリ入力です。セカンダリ入力のファイルへの参照には、プライマリから配布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: 入力タブ

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、 の下に少なくとも 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、「ソースの操作」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既に定義されている必要があります。

入力アーティファクトを指定しない場合は、 の下に少なくとも 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による作業。

> **Note**
>
> アーティファクト - 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) を検証するときに 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ことが原因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する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。引用符を使用して、変数名で特殊文字とスペースを有効に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変数の操作。

対応する UI: 入力タブ/可変 - 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、および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による作業。

対応する UI: タブ/アーティファクトを出力する

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、および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による作業。

対応する UI: タブ/Artifacts/Add artifact/Build artifact name を出力します。

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 が含まれる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以前のアクションのアーティファクトに存在でき、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による作業。

> **Note**
>
> 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: ビルドによって生成されたタブ/アーティファクト/アーティファクト/ファイルの追加を出力します。

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変数の操作。

対応する UI: タブ/可変/追加可変を出力します。

variable-name-1

(*action-name*/Outputs/Variables variable-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変数の操作。

対応する UI: タブ/可変/可変/名前を追加

AutoDiscoverReports

(###### /Outputs/AutoDiscoverReports)

(オプション)

自動検出機能の構成を定義します。

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 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、 はそれを CodeCatalyst レポート CodeCatalyst に変換します。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: タブ/レポート出力/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 レポートの前に CodeCatalyst を追加する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、 `TestSuiteOne.xml`と の 2 つのテストレポートを自動ディスカバートする場合`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: タブ/レポートを出力/レポートを自動的に検出/レポートプレフィックス

IncludePaths

(`######` /Outputs/AutoDiscoverReports/IncludePaths)

または

(*`action-name`*/Outputs/Reports/ *`report-name-1`*/IncludePaths)

(AutoDiscoverReportsが含まれて有効になっている場合、または Reports が含まれている場合に必須)

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、 を指定した場合`"/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ると、 CodeCatalyst 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すると、デフォルトは になります。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**
>
> 手動で設定されたレポートの場合、 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。

対応する UI:

- タブ/レポート/レポートを自動的に検出する reports/'Include/exclude paths'/Include paths
- タブ/レポート/レポートを手動で設定する reports/*report-name-1*/'Include/exclude paths'/Include paths

ExcludePaths

(###### /Outputs/AutoDiscoverReports/ExcludePaths)

または

(*action-name*/Outputs/Reports/ *report-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/**/*"、 /test/my-reports/ ディレクトリ内のファイルは検索 CodeCatalyst されません。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:

- タブ/レポート/レポートを自動的に検出する reports/'Include/exclude paths'/Exclude paths
- タブ/レポート/レポートを手動で設定する reports/*report-name-1*/'Include/exclude paths'/Exclude paths

SuccessCriteria

(###### /Outputs/AutoDiscoverReports/SuccessCriteria)

または

(*action-name*/Outputs/Reports/ *report-name-1*/SuccessCriteria)

(オプション)

テスト、コードカバレッジ、ソフトウェアコンポジション分析 (SCA)、および静的分析 (SA) レポートの成功基準を指定します。

詳細については、「レポートの達成基準の設定」を参照してください。

対応する UI:

- タブ/レポート/レポート/レポート/成功基準を自動的に検出する
- タブ/レポート/レポートを手動で設定する/*report-name-1*/成功基準を出力する

PassRate

(###### /Outputs/AutoDiscoverReports/SuccessCriteria/PassRate)

または

(*action-name*/Outputs/Reports/ *report-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。有効な値には小数が含まれます。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:

- タブ/レポート/レポートの自動検出/成功基準/合格率を出力する
- タブ/レポート/レポートを手動で設定する/*report-name-1*/成功基準/合格率を出力する

LineCoverage

(****** /Outputs/AutoDiscoverReports/SuccessCriteria/LineCoverage)

または

(*action-name*/Outputs/Reports/ *report-name-1*/SuccessCriteria/LineCoverage)

(オプション)

関連するレポートが合格としてマークされるためにカバーする必要があるコードカバレッジ CodeCatalyst レポート内の行の割合を指定します。有効な値には小数が含まれます。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:

- タブ/レポート/レポートを自動的に検出する/成功基準/明細カバレッジを出力する
- タブ/レポート/レポートを手動で設定する/*report-name-1*/成功基準/明細カバレッジを出力する

BranchCoverage

(****** /Outputs/AutoDiscoverReports/SuccessCriteria/BranchCoverage)

または

(*action-name*/Outputs/Reports/ *report-name-1*/SuccessCriteria/BranchCoverage)

(オプション)

関連するレポートが合格としてマークされるようにするためにカバーする必要がある、コードカバレッジ CodeCatalyst レポート内のブランチの割合を指定します。有効な値には小数が含まれます。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:

- タブ/レポート/レポートを自動的に検出する/成功基準/ブランチカバレッジを出力する
- タブ/レポート/レポートを手動で設定する/*report-name-1*/成功基準/ブランチカバレッジを出力する

Vulnerabilities

(###### /Outputs/AutoDiscoverReports/SuccessCriteria/Vulnerabilities)

または

(*action-name*/Outputs/Reports/ *report-name-1*/SuccessCriteria/Vulnerabilities)

(オプション)

関連付けられたレポートが合格としてマークされるように SCA CodeCatalyst 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最も低い値まで、、、`CRITICAL`、`HIGH`、`MEDIUM`、`LOW`、 ですINFORMATIONAL。

  例えば、 を選択すると`HIGH`、 `HIGH`および のCRITICAL脆弱性が増大します。
- 許可する指定された重要度の最大脆弱性数。この数を超えると、 CodeCatalyst レポートは失敗とマークされます。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・コンポジション分析レポート。

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テストレポートタイプ。

対応する UI:

- タブ/レポート/レポートを自動的に検出する/成功基準/脆弱性を出力する
- タブ/レポート/レポートを手動で設定する/*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: タブ/レポートを出力する

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: タブ/レポートを出力/レポート名を手動で設定

Format

(*action-name*/Outputs/Reports/ *report-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 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：
  - Clover XML の場合は、Clover (ビジュアルエディタ) または CLOVERXML (YAML エディタ) を指定します。
  - 「Coura XML」には、「Coura」 (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。
- ソフトウェア構成分析 (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: タブ/レポートを出力/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 Action の steps: セクションから CodeCatalyst ワークフローの Steps: セクションにコードを貼り付けます。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、#### でコードを削除し、太字 で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追加のコードで、 steps: セクション内に存在しない場合は、CodeCatalyst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定義リファレンス、 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、このガイドの範囲外です。

Actions GitHub 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

「Amazon ECS タスク定義をレンダリングする」アクションリファレンス

以下は、レンダリング Amazon ECS タスク定義アクションのアクション定義 YAML リファレンスです。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「Amazon ECS タスク定義をレンダリング」アクションの追加。

> **Note**
>
> 次の YAML プロパティのほとんどには、ビジュアルエディタに対応する UI 要素があります。UI 要素を検索するには、Ctrl+F を使用します。 要素は、関連する 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と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、「アクションバージョンでの作業」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には、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。

「依存関係」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アクションの設定。

対応する UI: での入力タブ/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。ワークフローレベルまたはアクションレベルでコンピューティングを指定できますが、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、ワークフローで定義されたすべてのアクションにコンピューティング設定が適用されます。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、「アクション間での計算の共有」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、「コンピュートタイプについて」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専用マシンのセットを設定します。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)

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 (YAML エディタ) 、または時間と分単位 (ビジュアルエディタ) で指定します。最小値は 5 分で、最大値は で説明

されていますのワークフローのクォータ CodeCatalyst。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: 入力タブ

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は必須)

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、「ソースの操作」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須)

デプロイするタスク定義ファイルが、以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による作業。

対応する UI: 設定タブ/Artifacts - 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する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。引用符を使用して、変数名で特殊文字とスペースを有効に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変数の操作。

対応する UI: 入力タブ/可変 - 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存在する場合、パスはソースリポジトリのルートフォルダからの相対パスです。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の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 フィールド、 `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)

レンダー 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 フィールド、`image`フィールドにこの名前を追加します。`image` 値が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`MyDockerImage:`*`commit-id`*タスク定義ファイルに を追加します。ここで *`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、*`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変数の操作。

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containerDefinitions、タスク定義ファイルの フィールド、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、他のアクションの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による作業。

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 にデプロイ 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ワークフローの例。

対応する UI: タブ/アーティファクト/名前を出力します

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/Artifacts/ファイル出力

Variables

(*ECSRenderTaskDefinition*/Outputs/Variables)

(必須)

レンダリングアクションで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task-definition-*random-string*.json) に設定します。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 にデプロイ アクションに追加する方法については、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: タブ/可変/名前フィールドを出力します

のワークフローのクォータ CodeCatalyst

次の表は、Amazon のワークフロー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

スペースあたりの最大ワークフロー数	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
1 スペースあたりのアクティブフリートの最大数	10
フリートあたりのアクティブなコンピュート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
1 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	500 文字以上。値を出力するアクションによって異なります。

	Note アクションの制限を超えると、値が切り捨てられることがあります。
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
1 アクションあたりの最大レポート数	50
テストレポート 1 つ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 1 つあたりの最大ファイル数	20,000
1 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
スタティック分析レポート 1 つあたりの最大ファイル数	20,000
1 スペースあたりの同時ワークフローの最大実行数	100
ワークフローあたりの最大アクション数	50
1 つのワークフローで同時に実行されるアクションの最大数	50
1 スペースあたりに同時に実行されるアクションの最大数	200
1 つの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タイムアウトは 8 時間です。 その他すべてのアクションのタイムアウトは 1 時間です。

AWS アカウント スペースごとに関連付けられる環境の最大数	5,000
1 アクションあたりの最大シークレット数	5
1 スペースあたりの最大シークレット数	500,000

の問題点 CodeCatalyst

では CodeCatalyst、機能、バグ、およびプロジェクトに関係するその他の作業を監視できます。各作業は、課題と呼ばれる個別の記録に保存されます。課題にタスクのチェックリストを追加することで、課題をより小さな目標に分割できます。各課題には説明、担当者、ステータス、その他のプロパティを設定でき、検索、グループ化、絞り込みが可能です。既定のビューを使用して課題を表示することも、カスタムのフィルタリング、並べ替え、または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。課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課題の概念。最初の号を作成する方法については、を参照してくださいでの課題の作成 CodeCatalyst。

課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 はプロジェクトで働いている開発者です。彼とプロジェクトメンバーのLi Juan、Mateo Jackson、Wang Xiulanは協力して、やるべき作業を決定します。彼と仲間の開発者たちは、ワン・シウランが率いるシンク・アップ・ミーティングを毎日開催しています。チームビューの 1 つに移動してボードを引き上げます。ビューを作成することで、ユーザーとチームは課題のフィルター、グループ化、ソートを保存して、指定した条件を満たす課題を簡単に表示できます。ビューには、担当者ごとにグループ化され、優先度別にソートされた課題が表示され、各開発者の最も重要な課題と課題のステータスが表示されます。Jorge は完了すべきタスクが割り当てられると、タスクごとに課題を作成して作業を計画します。課題を作成するとき、Jorge は適切な [ステータス]、[優先度]、[作業見積もり] の工数を選択できます。大きな課題の場合、Jorge は課題にタスクを追加して、作業をより小さな目標に分割します。Jorge は、すぐに開始する予定はないので、バックログなどのドラフトステータスで課題を作成します。ドラフトステータスの課題は [ドラフト] ビューに表示され、そこで計画と優先順位付けが行われます。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]、または [完了]) のステータスに更新して、対応する課題をボードに移動します。各タスクに取り組んでいる間、チームはタイトル、ステータス、担当者、ラベル、優先度、見積もりで絞り込んで、指定されたパラメータに一致する特定の課題や類似の課題を見つけることができます。Jorge と彼のチームはボードを使って課題ごとに完了したタスクの数を確認できるほか、day-to-day タスクが完了するまで各課題をあるステータスから次のステータスにドラッグして進捗状況を追跡できます。プロジェクトが進行するにつれて、完了した課題は [完了] ステータスに蓄積されます。Wang Xiulan は、開発者が現在および今後の作業に関連する課題に集中できるように、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ビューから削除することにしました。

作業を計画する際、プロジェクトに携わる開発者は [ソート基準] と [グループ化] を選択して、バックログからボードに移動したい課題を探します。最も優先度の高い顧客リクエストに基づいてボードに課題を追加することもあるでしょう。その場合は、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、達成可能な量の作業を引き受けているかどうかを確認するため

に、見積もり順に並べ替えることもあります。プロジェクトマネージャーの Saanvi Sarkar は、バックログを定期的に見直して整理し、プロジェクトの成功における各問題の重要性が優先順位に正確に反映されていることを確認します。

トピック

課題の概念

課題を作成すると、プロジェクト内で行われている作業を迅速かつ効率的に追跡できます。課題を活用して、毎日の同期会議で仕事を話し合ったり、仕事の優先順位を決めたり、その他にも活用できます。

このページには、で課題を効果的に活用するのに役立つ概念のリストが含まれています。CodeCatalyst

アクティブイシュー

アクティブな課題とは、ドラフトステータスにない、またはアーカイブされていない課題のことです。言い換えると、アクティブな課題とは、「未開始」、「開始」、「完了」のいずれかのステータスカテゴリのステータスを持つ課題です。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のカテゴリ。

デフォルトの [アクティブな課題] ビューでは、プロジェクト内のアクティブな課題をすべて表示できます。

アーカイブされた課題

アーカイブされた課題とは、プロジェクトに関係がなくなった課題です。たとえば、課題が完了して [完了] 列に表示する必要がなくなった場合や、誤って作成された課題をアーカイブできます。アーカイブされた課題は、必要に応じてアーカイブ解除できます。

担当者

担当者は、課題が割り当てられる担当者です。検索してもその人がリストに表示されない場合、その人はプロジェクトに追加されていません。追加する方法については、を参照してくださいユーザーをプロジェクトに招待する。。1 つの課題に複数の担当者を割り当てることができるようにするには、を参照してください。複数の担当者を有効または無効にする複数の担当者がいる課題は、それぞれが担当者の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、プロジェクト内の課題の追跡と管理に関するニーズに応じて、課題のさまざまな属性をカスタマイズできます。たとえば、ロードマップ用のフィールド、特定の期日、依頼者フィールドを追加できます。

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。見積もりには、問題のあいまいさや複雑さに加えて、必要な作業量も考慮に入れてください。リスク、難易度、未知数の問題が大きい問題には、より高い見積もりを使用してください。プロジェクトの見積もりタイプは変更できます。見積もり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。課題工数見積もりの設定

問題

課題とは、プロジェクトに関連する作業を追跡する記録です。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課題を作成できます。アジャイル開発を使用している場合は、課題にエピックやユーザーストーリーを説明することもできます。

ラベル

ラベルは課題のグループ化、ソート、フィルタリングに使用されます。新しいラベル名を入力するか、入力されたリストからラベルを 1 つ選択できます。このリストには、プロジェクトで最近使用されたラベルが含まれています。課題には複数のラベルを付けることができます。また、1 つのラベルを課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。

優先度

優先度とは、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のカテゴリ

ステータスは課題の現在の状態であり、問題の発生から完了までのライフサイクル全体にわたる進捗状況をすばやく確認するために使用されます。すべての課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、ステータスを整理したり、デフォルトの課題ビューに表示したりするのに役立ちます。

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で囲まれています:バックログ (ドラフト)、To Do (未開始)、進行中 (開始)、レビュー中 (開始)、Done (完了)。

ステータスの操作について詳しくは、を参照してください。ステータス

タスク

タスクを課題に追加して、その課題の作業をさらに細分化して整理することができます。作成時に課題にタスクを追加したり、既存の課題にタスクを追加したりできます。課題を表示しているときに、その課題のタスクを並べ替えたり、削除したり、完了のマークを付けたりできます。

ビュー

CodeCatalyst プロジェクト内の課題はビューに表示されます。ビューには、課題をリスト形式で表示するグリッドビューと、課題ステータス別に整理された列に課題をタイルとして表示するボードビューがあります。デフォルトビューは 4 つあり、グループ化、フィルタリング、ソートをカスタマイズして独自のビューを作成できます。以下のリストには、4 つのデフォルトビューの詳細が記載されています。

- ドラフトビューは、現在処理されていない課題を表示する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課題は、すべてこのビューに表示されます。チームはこのビューを使用して、どの課題がまだ定義されているか、または割り当てられて処理されるのを待っている課題を確認できます。
- アクティブな課題ビューは、現在処理中のすべての課題をボードビューで表示します。ステータスが「未開始」、「開始」、または「完了」のいずれかのステータスの課題は、このビューに表示されます。
- すべての課題ビューはグリッドビューで、ドラフトとアクティブな課題の両方、プロジェクト内のすべての課題が表示されます。
- アーカイブビューには、アーカイブされたすべての課題が表示されます。

での課題の作成 CodeCatalyst

Amazon CodeCatalyst のジェネレーティブ AI 機能はプレビューリリース中であり、変更される可能性があります。これらは米国西部 (オレゴン) リージョンでのみ利用できます。ジェネレーティブ AI 機能へのアクセスは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。

開発チームは課題を作成して、作業の追跡と管理に役立てます。ニーズに基づいてプロジェクト内で課題を作成できます。たとえば、コード内の変数の更新を追跡する課題を作成できます。課題をプロジェクト内の他のユーザーに割り当てたり、ラベルを使用して作業の追跡に役立てたりできます。

以下の手順に従って、で課題を作成します CodeCatalyst。

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. 課題を作成したいプロジェクトに移動します。
3. プロジェクトホームページで [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。Markdown を使用してフォーマットを追加できます。
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。プロジェクトの見積もり設定が [見積もりを非表示] に設定されている場合、見積もりフィールドは表示されません。
8. (オプション) 課題にタスクを追加します。タスクを使用して、課題の作業を小さな目標に分割できます。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タスクを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。

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、自分を担当者として簡単に追加できます。

> **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。

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。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。

12. [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。

Amazon Q に割り当てられた課題を作成および処理する際のベストプラクティス

課題を作成しても、課題の一部が残ってしまうことがあります。その原因は複雑でさまざまです。誰が取り組むべきかが明確でないことが原因の場合もあります。また、その課題がコードベースの特定の部分に関する調査や専門知識を必要とし、その作業に最適な候補者が他の問題で忙しい場合もあります。多くの場合、最初に取り組まなければならない緊急の仕事が他にもあります。これらの原因の一部またはすべてが、未解決の問題につながる可能性があります。 CodeCatalyst には、タイトルと説明に基づいて問題を分析できる Amazon Q と呼ばれるジェネレーティブ AI アシスタントとの統合が含まれています。問題を Amazon Q に割り当てると、Amazon Q は評価用のドラフトソリューションの作成を試みます。これにより、あなたとあなたのチームは注意が必要な問題に集中して作業を最適化することができます。一方、Amazon Q は、すぐに対処できるリソースがない問題の解決策に取り組みます。

> **Note**
>
> Amazon Bedrock を搭載: AWS 不正行為の自動検出機能を実装しています。課題を Amazon Q に割り当てる機能開発機能は 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、および人工知能 (AI) の責任ある使用を実現できます。

Amazon Q は、単純な問題と単純な問題で最も優れたパフォーマンスを発揮します。最良の結果を得るには、やりたいことを平易な言葉で明確に説明してください。Amazon Q が取り組めるように最適化された課題を作成するのに役立つベストプラクティスをいくつか紹介します。

- シンプルにしましょう。Amazon Q では、問題のタイトルと説明で説明できる単純なコード変更と修正が最も効果的です。タイトルがあいまいだったり、過度に華やかだったり、矛盾する説明で問題を割り当てたりしないでください。

- 具体的に記述してください。問題の解決に必要な正確な変更に関する情報が多ければ多いほど、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。可能であれば、変更したい API の名前、更新を希望するメソッド、変更が必要なテスト、その他考えられる詳細情報を記載してください。
- Amazon Q に問題を割り当てる前に、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できないため、Amazon Q に割り当てる前に、問題に必要な情報がすべて揃っていることを確認してください。
- コードの変更が必要な課題は、1 つのソースリポジトリにのみ割り当ててください。Amazon Q は、の 1 つのソースリポジトリ内のコードのみを処理できます CodeCatalyst。リンクされたリポジトリはサポートされていません。その課題を Amazon Q に割り当てる前に、その課題が 1 つのソースリポジトリ内の変更のみを必要とすることを確認してください。
- Amazon Q が推奨するデフォルトを使用して各ステップを承認します。デフォルトでは、Amazon Q は各ステップについてお客様の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に関するコメントでも Amazon Q とやり取りできるようになります。これにより、Amazon Q のアプローチを調整したり、作成するコードを改良したりして問題を解決するのに役立つ、よりインタラクティブな操作が可能になります。

  > **Note**
  >
  > Amazon Q は、Issue 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、アプローチの再考や改訂の作成を求められた場合は、レビューを行います。

- Amazon Q が提案するアプローチを常に注意深く確認してください。 こ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を生成する作業を開始します。Amazon Q に続行を指示する前に、アプローチが正しいと思われ、期待する詳細情報がすべて含まれていることを確認してください。
- レビュー前に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 がワークフローで動作することを許可するようにしてください。プロジェクトには、プルリクエストイベント時に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、Amazon Q が作成したワークフロー YAML の作成または更新を含むプルリクエストは、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が確実でない限り、Amazon Q にワークフローファイルの処理を許可することを選択しないでください。

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。

での課題の編集とコラボレーション CodeCatalyst

目次

課題を編集する

課題のタイトル、説明、ステータス、担当者、優先度、見積もり、ラベルを編集するには、次の手順に従います。

課題を編集するには

1. 編集する課題を選択し、課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索と表示。
2. 課題の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て 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、課題のタスクを表示および管理できます。詳細については、「課題のタスク管理」を参照してください。

5. ステータス、見積もり、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、既存のラベルを追加したり、新しいラベルを作成したり、ラベルを削除したりできます。

   a. 既存のラベルを追加するには、[+ Add label] 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、+ Add label を選択し、作成するラベルの名前を検索フィールドに入力して 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。すべての課題からラベルを削除すると、そのラベルは課題設定の「ラベル」セクションの「未使用のラベル」セクションに表示されます。フィルターを使用したり、課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リストの最後に表示されます。すべてのラベル (使用済みと未使用) とラベルが付いている課題の概要は、課題設定で確認できます。
7. 課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。+ Add me を選択すると、自分を担当者に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、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、「Link pull request (プルリクエストをリンク)」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。

   > **Tip**
   >
   > プルリクエストへのリンクを課題に追加したら、リンクされたプルリクエストのリストで 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、課題ボードとは別の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、そのプルリクエストを表示したり移動したりできるのは、そのプロジェクトのメンバーであるユーザーだけです。
10. (オプション) 既存のカスタムフィールドを追加して設定するか、新しいカスタムフィールドを作成するか、カスタムフィールドを削除します。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。

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。すべての課題からカスタムフィールドを削除すると、そのカスタムフィールドは削除され、フィルタリングしても表示されなくなります。

添付ファイルの使用

Issue CodeCatalyst に添付ファイルを追加して、関連ファイルに簡単にアクセスできるようにすることができます。課題の添付ファイルを管理するには、以下の手順に従います。

課題に追加された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**
>
> 問題の添付ファイルは、Amazon CodeCatalyst によってスキャンまたは分析されません。どのユーザーも、悪意のあるコードやコンテンツを含む可能性のある添付ファイルを問題に追加する可能性があります。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するベストプラクティスをユーザーが理解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、削除するには

1. 添付ファイルを管理したい課題を選択します。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索と表示。
2. 添付ファイルを追加するには、「ファイルをアップロード」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でファイルに移動し、選択します。[開く] を選択して添付ファイルとして追加します。添付ファイルの最大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。

- 以下の文字はファイル名には使用できません。
  - 制御文字:0x00-0x1fと 0x80-0x9f
  - 予約文字:/、?、<>、\、:、*、|、および "
  - Unix 予約ファイル名:と . . .
  - 末尾のピリオドとスペース
  - Windows で予約されているファイル名:CON, PRN, AUX, NUL, COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9, LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8, and LPT9
- 添付ファイル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 をコピーしたい添付ファイルの横にある省略記号メニューを選択し、「URL をコピー」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。

課題のタスク管理

タスクを課題に追加して、その課題の作業をさらに細分化、整理、追跡できます。

課題のタスクを管理するには

1. タスクを管理したい課題を選択します。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索と表示。
2. タスクでは、課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. タスクを完了済みとマークするには、そのタスクのチェックボックスを選択します。
   3. タスクの詳細を表示または更新するには、一覧からタスクを選択します。

4. タスクを並べ替えるには、タスクを選択してチェックボックスの左側からドラッグします。
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、「削除」を選択します。

課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します。

課題への取り組みを妨げているものがある場合は、その課題をブロック済みとしてマークするとよいでしょう。たとえば、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課題は、ブロックされる可能性があります。

課題をブロック済みとしてマークすると、その課題に赤い [ブロック済み] CodeCatalyst ラベルが追加され、バックログ、アーカイブ、またはボードで目立つようになります。

外部の状況が解決されたら、課題のブロックを解除できます。

課題をブロック済みとしてマークするには

1. ブロック済みとしてマークしたい課題を開きます。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索と表示。
2. [アクション] を選択し、[ブロック済みとしてマーク] を選択します。

課題のブロックを解除するには

1. ブロックを解除したい課題を開きます。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索と表示。
2. [アクション] を選択し、[ブロック解除済みとしてマーク] を選択します。

コメントの追加、編集、削除

課題には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、関連する課題、コードにタグを付けることができます。

課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 [課題] を選択します。
3. コメントを追加したい課題を選択します。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索と表示。

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

課題に対するコメントは編集できます。編集できるのは、自分が作成したコメントだけです。

1. プロジェクトに移動します。
2. ナビゲーションバーで [課題] を選択します。
3. コメントを編集したい課題を選択します。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索と表示。
4. コメントを編集するには、編集したいコメントを探します。

> **Tip**
>
> コメントは、古い順、または新しい順に並べ替えることができます。コメントは一度に 10 件読み込まれます。

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

課題に対するコメントは削除できます。削除できるのは、自分が作成したコメントだけです。コメントが削除されると、ユーザー名は表示されますが、元のコメントテキストの代わりに「このコメントは削除されました」と表示されます。

1. プロジェクトに移動します。
2. ナビゲーションバーで [課題] を選択します。
3. コメントを削除したい課題を選択します。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索と表示。
4. 省略記号アイコンを選択し、[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、関連する課題、コメント内のコードをメンションできます。そうすることで、メンション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。

コメントの @mention に

1. プロジェクトに移動します。
2. ナビゲーションバーで [課題] を選択します。
3. 編集する課題を選択し、課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索と表示。
4. 「コメントを追加」テキストボックスを選択します。
5. @*user_name*入力して別のユーザーをメンションします。
6. @*project_name*プロジェクトを入力してメンションする。
7. @*issue_name*@*issue_number*または別の課題について言及する。
8. @*file_name*ソースリポジトリ内の特定のファイルまたはコードを入力して言及します。

   > **Note**
   >
   > 入力した内容と一致する用語を含む上位 5 つの項目 (ユーザ、ソースリポジトリ、プロジェクトなど) @mention のリストが表示されます。

9. メンションしたい項目を選択します。アイテムがどこにあるかを示す経路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。

課題の進行中

すべての課題にはライフサイクルがあります。では CodeCatalyst、課題は通常、バックログのドラフトとして開始されます。その課題の作業が開始されると、その課題は別のステータスカテゴリに移動し、完了するまでさまざまなステータスを経てからアーカイブされます。次の方法で、課題をライフサイクル全体にわたって移動または進行させることができます。

- 課題をバックログとボード間で移動できます。
- 進行中の課題をさまざまな完了ステージに移動できます。

- 完了した課題はアーカイブできます。

バックログとボードの間

課題の作業を開始したら、課題をバックログからボードに移動できます。作業が延期された場合は、課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課題を移動するには

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 [課題] を選択します。デフォルトビューはボードです。
3. 課題をボードからバックログに移動するには:
    a. 移動したい課題を選択します。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索と表示。
    b. ドロップダウンの [ステータス] メニューから [バックログ] を選択します。
4. 課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. 移動したい課題を選択します。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索と表示。
    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。

ボード上で課題をライフサイクルの各段階に進めます。

ボード内の課題は、完了するまでさまざまなステータスに移動できます。

ボード内で課題を移動するには

1. ナビゲーションペインで [課題] を選択します。デフォルトビューはボードです。
2. 次のいずれかを行います。
    - 課題を別のステータスにドラッグアンドドロップします。
    - 課題を選択し、「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。
    - 課題を選択し、[移動先:*next-status*] を選択します。

課題のアーカイブについて詳しくは、を参照してください。課題をアーカイブする。

グループ間での課題の移動

[すべての課題] ビューと [ボード] ビューでは、課題をさまざまなパラメータでグループ化できます。課題がグループ化されている場合は、課題をあるグループから別のグループに移動できます。課題をあるグループから別のグループに移動すると、課題がグループ化されたフィールドがターゲットグループに合わせて自動的に編集されます。

シナリオの例として、Wang Xiulan と CodeCatalyst Saanvi Sarkar という 2 人の担当者に課題が割り当てられている会社があるとします。ボードはによってグループ化されており、担当者にそれぞれ 1 つずつ`Assignee`、合計 2 つのグループがあります。課題を Wang Xiulan グループから Saanvi Sarkar グループに移動すると、課題の担当者は Saanvi Sarkar に更新されます。

課題をアーカイブする。

> **Note**
>
> 課題はプロジェクト内で削除されるのではなく、アーカイブされます。課題を削除するには、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課題はアーカイブできます。課題をアーカイブすると、 CodeCatalyst アーカイブされた課題を除外するすべてのビューから課題が削除されます。アーカイブされた課題は、アーカイブされた課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を解除できます。

課題は次の場合にアーカイブします。

- 課題が完成し、「完了」列にその課題が不要になった。
- この課題に取り組む予定はありません。
- 間違って作成しました。
- アクティブな課題の数が最大数に達しました。

課題をアーカイブするには

1. アーカイブしたい課題を開きます。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索と表示。

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) ステータスが「完了」の複数の課題をすばやくアーカイブするには、ボード上のいずれかの完了ステータスの上部にある縦の省略記号を選択し、「課題をアーカイブ」を選択します。

課題をアーカイブ解除するには

1. アーカイブ解除したい課題を開きます。課題ビュースイッチャーのドロップダウンメニューから「アーカイブされた課題」ビューを開くと、アーカイブされた課題のリストを表示できます。問題を見つける方法については、を参照してください。問題の検索と表示
2. [アーカイブ解除] を選択します。

問題の検索と表示

以下のセクションでは、 CodeCatalyst プロジェクト内の課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。

CodeCatalyst プロジェクト内の課題はビューに表示されます。ビューには、課題をリスト形式で表示するグリッドビューと、課題ステータス別に整理された列に課題をタイルとして表示するボードビューがあります。デフォルトビューは 4 つあり、グループ化、フィルタリング、ソートをカスタマイズ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課題を表示する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課題は、すべてこのビューに表示されます。チームはこのビューを使用して、どの課題がまだ定義されているか、または割り当てられて処理されるのを待っている課題を確認できます。
- アクティブな課題ビューは、現在処理中のすべての課題をボードビューで表示します。ステータスが「未開始」、「開始」、または「完了」のいずれかのステータスの課題は、このビューに表示されます。
- すべての課題ビューはグリッドビューで、ドラフトとアクティブな課題の両方、プロジェクト内のすべての課題が表示されます。
- アーカイブビューには、アーカイブされたすべての課題が表示されます。

課題を検索する

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。検索を絞り込む方法の詳細については、を参照してください[で検索 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、「[で検索 CodeCatalyst]」を参照してください。

ソートに関する問題

デフォルトでは、 CodeCatalyst の課題は手動の順序でソートされます。手動順序では、課題はユーザーが移動した順序で表示されます。手動順序で並べ替えられた課題をドラッグアンドドロップして順序を変更できます。この並べ替えオプションは、課題のバックログを整理したり、課題に優先順位を付けたりする場合に役立ちます。

以下の表は、グリッドビューとボードビューの両方で課題をソートする方法を示しています。

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
見積もり	見積もり
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	

課題のソート方法を変更するには、以下の手順に従います。

課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 [Issues] を選択します。デフォルトビューはボードです。
3. (オプション)「アクティブな課題」を選択して課題ビュースイッチャーのドロップダウンメニューを開き、別の課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. ソートの基準に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. 「Sort by」ドロップダウンメニューを選択し、ソートの基準となるパラメータを選択します。課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、ソートの基準となるパラメータを選択します。課題は昇順でソートされます。

課題をグループ化します。

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でボード上の課題を整理するために使用されます。

課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 [課題] を選択します。デフォルトビューはボードです。
3. (オプション)「アクティブな課題」を選択して課題ビュースイッチャーのドロップダウンメニューを開き、別の課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、ラベルを選択してから [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、現在課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。課題は、設定したパラメータと一致するグループにのみ表示されます。

フィルターの問題

フィルターを使用して、指定した名前、優先度、ラベル、カスタムフィールド、または担当者を含む課題を検索します。

課題を絞り込むには

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. (オプション) [アクティブな課題] を選択して課題ビュースイッチャードロップダウンメニューを開き、別の課題ビューに移動します。

> **Note**
>
> 課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、課題検索バーにその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. フィルターするパラメーターを選択します。複数のフィルターとパラメーターを選択できます。and または or を選択することで、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課題が表示されるようにフィルターを設定できます。ビューが更新され、フィルターに一致する課題が表示されます。

課題ビューを作成する

ビューを作成すると、特定のフィルターセットに一致する課題をすばやく表示できます。これにより、時間を節約でき、以前にフィルター、グループ化、または並べ替えを行った課題をすばやく表示できます。

課題ビューを作成するには

1. ナビゲーションペインで [Issues] を選択します。

2. (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したい場合があります。別のビューに移動するには、[アクティブな課題] を選択して課題ビュー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、並べ替えを設定します。これらはビューの作成中に追加できますが、以前に追加したことがある場合は、ビューを作成する前にビューに表示されている内容をプレビューできます。
4. ヘッダーバーから課題ビュースイッチャーのドロップダウンメニューを開きます。課題がステータスに基づいて列に表示されるボードビューを作成するには、ボード列の [+] を選択します。課題を一覧表示するグリッドビューを作成するには、グリッド列の [+] を選択します。気が変わったら、ビューの種類は作成前に変更できます。
5. 「ビューを作成」ダイアログで、ビューの名前を入力します。
6. [フィルター]、[課題のグループ化]、[課題の並べ替え] の各フィールドは、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てください。
7. [ビューを作成] を選択してビューを作成し、そのビューに切り替えます。

エクスポートに関する問題

現在のビューにある課題を.xlsx ファイルにエクスポートできます。課題を エクスポートするには、次の手順を実行します。

課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 [課題] を選択します。
3. [アクティブな課題] を選択して課題ビュースイッチャードロップダウンメニューを開き、エクスポートしたい課題を含むビューに移動します。ビューに表示されている課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 ファイルがダウンロードされます。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がタイトルになっています。

課題設定の構成

以下のトピックでは、で課題の設定を行う方法について詳しく説明します CodeCatalyst。

トピック

複数の担当者を有効または無効にする

で課題を担当する複数の担当者の設定を構成するには、次の手順に従います。 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 [Issues] を選択します。
2. [アクティブな課題] を選択して課題ビュー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、インジケーターを切り替えて複数の担当者を同じ課題に割り当てられるようにします。1 つの課題には最大 10 人の担当者を割り当てることができます。このオプションを有効にしない場合、1 つの課題に割り当てることができる担当者は 1 人だけです。

課題工数見積もりの設定

次の手順に従って、で課題の工数見積もりの設定を行います。 CodeCatalyst

課題の工数見積もりを設定するには

1. ナビゲーションペインで [Issues] を選択します。
2. [アクティブな課題] を選択して課題ビュー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 [見積もり] で、見積もり値の表示方法を選択します。推定値のタイプは、T シャツサイズ、フィボナッチシーケンシング、または推定値の非表示です。見積もりタイプが更新されても、データが失われることはなく、すべての案件の見積額が自動的に換算されます。変換マッピングを以下の表に示します。

T シャツサイズ	フィボナッチシーケンス
XS	1
XS	2
S	3
M	5
L	8
XL	13

ステータス

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]、または [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へ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、を参照してくださいステータスとステータスのカテゴリ。ビューの詳細については、を参照してください。問題の検索と表示

ステータスを作成するには

1. ナビゲーションペインで [Issues] を選択します。
2. [アクティブな課題] を選択して課題ビュー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、ステータスを設定したい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**
>
> X アイコンを選択すると、ステータスの追加をキャンセルできます。

カスタムステータスがボードに表示され、課題の作成時にオプションとして表示されるようになりました。

ステータスを編集するには

1. ナビゲーションペインで [Issues] を選択します。
2. [アクティブな課題] を選択して課題ビュー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、編集または変更したい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。

   編集したステータスがボードに表示されるようになりました。

ステータスを移動するには

1. ナビゲーションペインで [Issues] 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、移動したいステータスを選択します。
4. ステータスを目的の場所にドラッグアンドドロップします。

> **Note**
>
> ステータスは指定されたカテゴリ内でのみ移動できます。

   これで、ボード上のステータスの順序が変更されました。

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 [Issues] を選択します。
2. [アクティブな課題] を選択して課題ビュー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、無効にするステータスを選択します。
4. 非アクティブ化したいステータスで、その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になりました。

> **Note**
>
> すべての課題がボードから移動されるまで、非アクティブ化されたステータスはボードに表示されます。課題を非アクティブ化されたステータスに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度有効にするには、ステータスのトグルを選択します。ステータスはもうグレーアウトされません。

> **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、そのステータスを無効にすることはできません。

ラベル

課題のラベルをカスタマイズできます。これには、ラベルの編集や色の変更が含まれます。ラベルは作業の分類や整理に役立ちます。

ラベルを作成する

では CodeCatalyst、新しい課題を作成するとき、または既存の課題を編集するときにラベルを追加して作成します。詳細については、「での課題の作成 CodeCatalyst」および「での課題の編集とコラボレーション CodeCatalyst」を参照してください。

ラベルを編集する。

以下の手順に従って、既存のラベルの名前または色を変更します。

ラベルを編集するには

1. ナビゲーションペインで [Issues] を選択します。
2. [アクティブな課題] を選択して課題ビュー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てい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、カラー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. 変更したラベルが、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課題の数も確認できます。

> **Note**
>
> 各ラベルの横に表示されてい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課題を確認できます。

ラベルを削除する。

現在、では課題ラベルを削除できません CodeCatalyst。すべての課題からラベルを削除すると、そのラベルは課題設定の「ラベル」セクションの「未使用のラベル」セクションに表示されます。フィルターを使用したり、課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リストの最後に表示されます。すべてのラベル (使用済みと未使用) とラベルが付いている課題の概要は、課題設定で確認できます。

カスタム フィールド

カスタムフィールドを作成すると、プロジェクトの作業を整理して確認しやすくなります。カスタムフィールドはフィルターの使用可能なフィルターのリストに追加され、カスタムフィールドで課題を絞り込むことができます。カスタムフィールドは名前と値の組み合わせです。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。

課題には複数のカスタムフィールドを設定できます。

カスタムフィールドを作成する。

では CodeCatalyst、課題の作成時または既存の課題の編集時にカスタムフィールドを追加することでカスタムフィールドを作成します。詳細については、「での課題の作成 CodeCatalyst」および「での課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を削除する。

カスタムフィールドを削除する場合は、追加された各課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を削除すると、そのカスタムフィールドはフィルターに表示されなく

なります。フィルターを使用してカスタムフィールドを含むすべての課題を表示し、課題を編集することで課題を削除できます。詳細については、「問題の検索と表示」および「課題を編集する」を参照してください。

添付ファイルの表示と管理

課題設定で、プロジェクト内の課題に追加されたすべての添付ファイルを含む表を表示できます。このテーブルには、コンテンツタイプ、追加された日時、追加された課題とステータス、ファイルサイズなどの情報を含む、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、完成した課題やアーカイブされた課題のサイズの大きい添付ファイルを簡単に見分け、それらを削除して空き容量を増やすことができます。

> **Important**
>
> 問題の添付ファイルは Amazon CodeCatalyst によってスキャンまたは分析されません。どのユーザーも、悪意のあるコードやコンテンツを含む可能性のある添付ファイルを問題に追加する可能性があります。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するベストプラクティスをユーザーが理解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課題添付ファイルを表示して管理するには

1. ナビゲーションペインで [Issues] 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。

の問題のクォータ CodeCatalyst

次の表は、Amazon の問題の割り当てと制限を示しています。 CodeCatalystAmazon 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
進行中の課題	1 プロジェクトあたり最大 1,000 件。

リソース	デフォルトのクォータ
添付ファイルのサイズ	添付ファイルあたり最大 500 MB です。 添付ファイルの最大合計ストレージは、スペースの全体的な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。
問題の総数 (処理中およびアーカイブ済み)	1 プロジェクトあたり最大 100,000 件。
保存されたビュー	1 つのプロジェクトにつき最大 50 件の課題ビューを保存できます。
1 つの課題にリンクできるプルリクエストの数	1 つの課題につき最大 50 件のプルリクエスト。
ステータス (プロジェクトあたり)	1 プロジェクトあたり最大 50 個。
ステータス (課題あたり)	1 号あたり最大 50 件です。
ラベル (1 プロジェクトあたり)	1 プロジェクトあたり最大 200 個。
ラベル (1 冊あたり)	1 号あたり最大 50 個。
カスタムフィールド (1 件あたり)	1 号あたり最大 50 個です。
担当者	1 件あたり最大 10 件です。
コメント	1 号あたり最大 1,000 件。
タスク	1 号あたり最大 100 件です。

の ID、権限、アクセス CodeCatalyst

Amazon CodeCatalyst に初めてサインインするときは、 AWS ビルダー ID を作成します。 AWS ビルダー ID はには存在しません AWS Identity and Access Management。初回サインイン時に選択したユーザー名が、ID 固有のユーザー 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ログインできます。

- スペース作成の一環として。
- CodeCatalystでのプロジェクトやスペースへの招待を受け入れる一環として。

自分の ID に関連付けられている 1 つまたは複数の役割によって、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固有のものなので、あるプロジェクトではあるロールを、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、その ID CodeCatalyst がスペースに追加され、制限付きアクセスロールが割り当てられます。ユーザーをプロジェクトに招待するときは、プロジェクト内でユーザーに与えたいロールを選択します。これにより、ユーザーがプロジェクト内で実行できるアクションと実行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寄稿者の役割だけを必要とします。詳細については、「Amazon のロールを使った作業 CodeCatalyst」を参照してください。

Git クライアントまたは統合開発環境 (IDE) を使用する場合、プロジェクト内のユーザーがプロジェクトのソースリポジトリにアクセスするには、プロジェクトロールに加えて、個人アクセストークン (PAT) が必要です。プロジェクトメンバーは、この PAT を ID 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サードパーティアプリケーションで使用できます。 CodeCatalyst たとえば、ソースリポジトリをローカルコンピューターに複製する場合、PAT とユーザー名を指定する必要があります。 CodeCatalyst

CodeCatalyst AWS とリソース間のアクセスを設定するには、ワークフローにアクションをデプロイするときに、AWS CloudFormation スタックやリソースへのアクセスなどのアクションを実行するサービスロールを使用します。プロジェクトテンプレートに含まれているワークフローアクションを実行するには、 CodeCatalyst AWS と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

- Amazon のロールを使った作業 CodeCatalyst
- Amazon での個人アクセストークンの管理 CodeCatalyst

- Amazon での多要素認証 (MFA) CodeCatalyst
- Amazon ンのセキュリティ CodeCatalyst
- Amazon でのモニタリング CodeCatalyst
- の ID、許可、アクセスのクォータ CodeCatalyst
- トラブルシューティング

Amazon のロールを使った作業 CodeCatalyst

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、ロールによって、そのプロジェクトのリソースを使ってユーザーがプロジェクト内で実行できることが指定されます。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になります。スペースの管理者としてユーザーを追加または削除できます。スペース管理者ロールには、のどのロールよりも幅広い権限があります。 CodeCatalystベストプラクティスとして、各自の業務に必要な最も狭い権限をユーザーに割り当ててください。

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。メンバーがメンバーになってい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 1 つのプロジェクトまたはスペースで 1 つのロールしか持つことができませんが、ユーザーはプロジェクトやスペースごとに異なるロールを持つことができます。たとえば、あるユーザーがあるプロジェクトではプロジェクト管理者ロールを、別のプロジェクトでは投稿者ロールを持つ場合があります。

トピック

- ロールタイプ
- 各ロールで使用できる権限
- ユーザーロールの表示と変更

ロールタイプ

スペースには次の 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承諾したユーザーには、その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 つのロールがあります。

- プロジェクト管理者
- 寄稿者
- レビュー担当者
- 読み取り専用

ユーザーをプロジェクトに追加すると、 CodeCatalyst そのユーザーには自動的に制限付きアクセス権限が付与されます。すべてのプロジェクトからユーザーを削除すると、 CodeCatalystそのユーザーから制限付きアクセスロールが自動的に削除されます。

スペース管理者ロール

CodeCatalystスペース管理者ロールはで最も強力なロールです。スペース管理者ロールにはすべての権限があるため、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スペース管理者ロールを割り当ててください。 CodeCatalystスペース管理者ロールを持つユーザーは、スペース管理者ロールから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、 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、元のスペース作成者が不在の場合に備えて、このロールを担当できる他のユーザーにこのロールを少なくとも 1 人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で 2 番目に強力なロールですが、 CodeCatalystスペース内のプロジェクトにはアクセスできません。スペース内でプロジェクトを作成し、スペースのユーザーとリソースの管理を支援する必要があるユーザー向けに設計されています。パワーユーザーロールは、作業の一環としてスペース内でプロジェクトを作成したりユーザーを管理したりする必要のあるチームリーダーまたはマネージャーのユーザーに割り当てます。

制限付きアクセスロール

制限付きアクセスロールは、 CodeCatalyst スペース内でほとんどのユーザーが持つロールです。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の作業に必要な限定的な権限が付与されま

す。制限付きアクセスロールは、そのスペースに直接招待したユーザーに割り当てます。ただし、そのユーザーの仕事でスペースの一部を管理する必要がある場合を除きます。

プロジェクト管理者ロール

CodeCatalyst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。このロールは、プロジェクト設定の編集、プロジェクト権限の管理、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てください。

プロジェクトロールにはスペースレベルでの権限はありません。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。スペース管理者ロールまたはパワー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**
>
> スペース管理者ロールにはすべての権限があります CodeCatalyst。

コントリビューターロール

コントリビューターロールは、プロジェクトに参加する大多数のメンバーを対象としています。CodeCatalyst このロールは、プロジェクト内のコード、ワークフロー、課題、アクションを扱う必要があるユーザーに割り当ててください。

レビュー担当者の役割

レビュー担当者ロールは、プルリクエストや Issue など、プロジェクト内のリソースを操作する必要はあるが、コードの作成やマージ、ワークフローの作成、プロジェクト内のワークフローの実行の開始や停止はできないユーザーを対象としています。 CodeCatalyst プルリクエストの承認とコメント、課題の作成、更新、解決、コメントのほか、プロジェクト内のコードとワークフローの表示を行う必要があるユーザーには、レビュー担当者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、リソースやリソースのステータスを確認する必要はあるが、リソースを操作したり、プロジェクトに直接貢献したりする必要は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、リソースを内で作成することはできませんが CodeCatalyst、リポジトリを複製したり、Issue の添付ファイルをローカルコンピュータにダウンロードしたりするなど、リソースを表示したりコピーすることはできます。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、直接操作する必要はないユーザーには、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権限

次の表は、 CodeCatalyst 各ロールで使用できる権限を示しています。リンクを使用して適切な権限セットに移動してください。

- 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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
スペース権限							
スペースを作成	✓	✗	✗	✗	✗	✗	✗
スペース請求の詳細を編集する	✓	✗	✗	✗	✗	✗	✗
シングルサインオンを設定	✓	✗	✗	✗	✗	✗	✗

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	コントリビューターロール	レビュー担当者の役割	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	✓	✗	✗	✗	✗	✗	✗
スペースでジェネレーティブ AI 機能を有効にする	✓	✗	✗	✗	✗	✗	✗
スペースのジェネレーティブ AI 機能を無効にする	✓	✗	✗	✗	✗	✗	✗
スペースを削除	✓	✗	✗	✗	✗	✗	✗
他のユーザーをスペース管理者ロールに追加します。	✓	✗	✗	✗	✗	✗	✗

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
スペース管理者ロールから他のユーザーを削除する							
チームを作成							
チームを削除する							
チームを更新							
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します。							
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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
AWS アカウント接続をスペースに関連付ける	✓	✓	✗	✗	✗	✗	✗
AWS アカウント接続を更新する	✓	✓	✗	✗	✗	✗	✗
AWS アカウント接続とスペース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	✓	✓	✗	✗	✗	✗	✗

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
他のユーザーをスペースに招待します。							
VPC 接続の作成							
VPC 接続を編集							
VPC 接続を削除							
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 CodeCatalyst							

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
スペースを表示							
チームを表示							
VPC 接続を表示する							
拡張機能、権限	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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
拡張機能のインストール							
拡張機能の更新							
拡張機能の削除							
GitHub アカウントConnect する							
アカウントを接続解除する GitHub							

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
Jira サイト Connect する	✓	✓	✗	✗	✗	✗	✗
Jira サイトとの接続を切断する	✓	✓	✗	✗	✗	✗	✗
インストール済みの拡張機能の構成の詳細を表示する	✓	✓	✗	✗	✗	✗	✗
拡張機能を表示する	✓	✓	✓	✓	✓	✓	✓

プロジェクト権限	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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
プロジェクト設定を編集する	✓	✗	✗	✓	✗	✗	✗

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
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します。							
プロジェクトを削除する							
ユーザーをプロジェクトに招待する							
プロジェクト内のユーザーの役割を変更							

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
プロジェクトからユーザーを削除する	✓	✗	✗	✓	✗	✗	✗
プロジェクトをチームに追加する	✓	✗	✗	✓	✗	✗	✗
プロジェクトからチームを削除する	✓	✗	✗	✓	✗	✗	✗
チームのプロジェクトロールを変更	✓	✗	✗	✓	✗	✗	✗
プロジェクトを表示	✓	✗	✗	✓	✓	✓	✓
プロジェクトアクティビティを表示する	✓	✗	✗	✓	✓	✓	✓

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
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権限	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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
リポジトリを作成する							
リポジトリをリンクする							
リポジトリをリンク解除							
リポジトリを削除							
リポジトリ設定を編集する							

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
リポジトリを表示する							
リポジトリ設定を表示する							
リポジトリをクローン							
ブランチを作成する							
ブランチルールを作成する							
デフォルトブランチを変更							
ブランチを削除する							
ブランチをマージ							
ブランチのルールを更新							

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
ブランチを表示する							
ブランチのルールを表示する							
フォルダーの作成							
フォルダーの削除							
フォルダーの編集							
フォルダーの表示							
[ファイルを作成]							
ファイルを削除							
ファイルを編集する							

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
ファイルを表示する							
コミットの作成とプッシュ							
コミットを表示する							
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							
プルリクエストを更新							

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
プルリクエストの承認ルールを更新							
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							
プルリクエストをクローズする							
プルリクエストを承認する							
プルリクエストにコメントする							

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
プルリクエストのコメントでAmazon Q とやり取りする							
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							
課題をプルリクエストにリンクする。							
プルリクエストから課題をリンク解除する							

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
開発環境権限	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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
独自の開発環境を作成							
独自の開発環境を停止してください							
他のユーザーが作成した開発環境を停止します。							
独自の開発環境を再開する							
独自の開発環境を表示							

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	コントリビューターロール	レビュー担当者の役割	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表示する							
独自の開発環境を編集する							
他のユーザーが作成した開発環境を編集する							
自分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							
開発環境用の開発ファイルを作成します。							

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
開発環境用の devfile を編集します。							
開発環境用の開発ファイルを削除します。							
開発環境用の開発ファイルを表示する							

Package リポジトリとパッケージ権限	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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
パッケージリポジトリの作成							
パッケージリポジトリを表示する							

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
パッケージリポジトリを編集							
パッケージリポジトリを削除							
ゲートウェイパッケージリポジトリの作成							
ゲートウェイパッケージリポジトリを表示する							
ゲートウェイパッケージリポジトリを削除							

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
上流パッケージリポジトリの追加							
上流リポジトリの検索順序を編集							
上流のパッケージリポジトリを削除							
パッケージリポジトリ Connect する							
パッケージリポジトリからパッケージを読み込む							

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
パッケージをパッケージリポジトリに公開する							
上流のリポジトリからパッケージを読み込んで保持する							
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アセットを表示する							

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
パッケージバージョン依存関係を一覧表示する							
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							
パッケージバージョンを削除							

ワークフロー権限	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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
ワークフローを作成							

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
ワークフローを更新							
ワークフローを削除							
ワークフローを開始する							
ワークフローを停止							
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の更新							
ワークフローシークレットの削除							

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
環境の作成							
環境を削除する							
フリートを作成							
フリートを更新							
フリートを削除							
他のアカウントのワークフローリソースを管理							
VPC 接続を環境に関連付ける							
VPC 接続と環境と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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
VPC 接続環境をワークフローに関連付ける							
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							

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
テストアクションログを表示する	✓	✗	✗	✓	✓	✓	✓
ワークフローを表示する	✓	✗	✗	✓	✓	✓	✓
ワークフロー実行を表示	✓	✗	✗	✓	✓	✓	✓
ワークフローの実行結果を表示する	✓	✗	✗	✓	✓	✓	✓
ワークフローシークレットを表示する	✓	✗	✗	✓	✓	✓	✓
イシューと権限	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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
課題を作成	✓	✗	✗	✓	✓	✓	✗

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
課題を更新							
課題を表示							
課題をアーカイブする							
Amazon Q に課題を割り当てる							
問題に関するコメントで Amazon Q とやり取りする							
Amazon Q を課題から割り当て解除する							
他のユーザーが作成した課題を更新							

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
課題に関するコメントを表示する							
課題へのコメントの作成							
課題へのコメントを更新する							
ラベルの作成							
ラベルを更新する							
ラベルを表示する							
課題にラベルを追加する							
課題からラベルを削除する							

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
課題のカスタムステータスを作成							
カスタムステータスを更新する							
カスタムステータスを表示する							
カスタムステータスを移動する							
カスタムステータスを無効にする							
課題への添付ファイルの追加							

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
課題の添付ファイルを表示する							
課題から添付ファイルを削除する							
プルリクエストを課題にリンクする							
プルリクエストを課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解除							

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
カスタムブループリント権限	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	コントリビューターロール	レビュー担当者の役割	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							
カスタムブループリントのプレビューを公開します。							
プレビューカスタムブループリントを非公開にする							
カスタムブループリントを公開する							

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
カスタムブループリントを非公開にする							
カスタムブループリントをスペースブループリントカタログに追加							
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権限を管理します。							

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
カスタムブループリントのカタログバージョンを管理します。							
カスタムブループリントの更新							
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							

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
カスタムブループリントをプロジェクトから関連付け解除する							
適用し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します。							
公開済みのカスタムブループリントを表示する							

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
通知、権限	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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
通知チャネルを設定							
通知チャネルを削除							
通知設定を編集する							
通知設定を表示する							
CodeCatalyst インシデントに関する通知を自動的に受け取る							

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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
関連するメールアカウントのメール通知を設定します。							

検索権限	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	コントリビューターロール	レビュー担当者の役割	読み取り専用ロール
プロジェクト内を検索する							
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、プロジェクトでユーザーが実行できるアクションを理解しやすくなります。また、追加の権限が必要な場合は役割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役割を確認するには

1. 各プロジェクトメンバーに関連付けられているロールを確認したい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」タブの「役割」には、各プロジェクトメンバーの役割が表示されます。

プロジェクト内のユーザーの役割を変更するには

1. プロジェクトメンバーに関連付けられているロールを変更したい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」タブの「プロジェクトメンバー」で、役割を変更したい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。

スペース内のロールの表示と変更

CodeCatalyst でプロジェクトへの招待を受け入れたすべてのユーザーがプロジェクトのスペースのメンバーになります。スペースメンバーのリストを表示できます。ユーザーの役割を制限付きアクセスからスペース管理者に変更して、スペースとリソースをより適切に管理できます。スペース管理者ロールは、 CodeCatalystユーザがでプロジェクトを作成できる唯一のロールです。

> **Warning**
>
> CodeCatalystスペース管理者ロールはで最も強力なロールです。このロールを持つユーザは、スペースの削除など CodeCatalyst、あらゆる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てください。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のロールを変更するには

1. https://codecatalyst.aws/ CodeCatalyst で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、[Change role] を選択します。
5. 「役割の変更」で、割り当てる役割を選択し、「確認」を選択します。

Amazon での個人アクセストークンの管理 CodeCatalyst

Git クライアントまたは統合開発環境 (IDE) CodeCatalyst を使用してローカルコンピューター上のソースリポジトリなどの一部の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 PAT は、内のすべてのスペースとプロジェクトのユーザー ID に関連付けられます。CodeCatalystID には複数の PAT を作成できます。 CodeCatalyst

作成した PAT の名前と有効期限を確認したり、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、PAT の有効期限は 1 年です。

PAT の作成

PAT は内のユーザ ID と関連付けられます。 CodeCatalystPAT シークレットは、作成時にのみコピーできます。

PAT の作成 (コンソール)

コンソールを使用して PAT を作成できます。 CodeCatalyst

個人アクセストークンを作成するには (コンソール)

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. [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、[Copy] を選択します。PAT シークレットは、取得できる場所に保存します。

> **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は取得できません。PAT シークレットを安全な場所に保存していない場合は、別の PAT シークレットを作成できます。

.

パットの作成 (CLI)

CLI を使用してで PAT を作成できます。 CodeCatalyst

個人アクセストークン ()AWS CLIを作成するには

1. ターミナルまたはコマンドラインで、create-access-token以下のようにコマンドを実行します。

   ```
   aws codecatalyst create-access-token
   ```

   成功すると、以下の例のように、作成された PAT に関する情報が返されます。

   ```
   {
       "secret": "value",
       "name": "marymajor-22222EXAMPLE",
       "expiresTime": "2024-02-04T01:56:04.402000+00:00"
   }
   ```

2.

PAT シークレットは PAT を作成したときに 1 回しか表示できません。PAT シークレットを置き忘れた場合や、安全に保存されていないことが心配な場合は、別の PAT シークレットを作成できます。

ユーザアカウントに関連付けられている PAT は、を使用して表示できます。 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は表示できません。

> **Note**
>
> AWS CLI 最新バージョンのを使用して作業していることを確認してください。 CodeCatalyst CodeCatalyst 以前のバージョンにはコマンドが含まれていない可能性があります。で使用する前に、 AWS CLI プロファイルを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。

PAT を表示する。

で PAT を表示できます。 CodeCatalystリストには、ユーザ ID に関連付けたすべての PAT が表示されます。PAT は、内のすべてのスペースとプロジェクトの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 PAT は有効期限が切れると削除されるため表示されません。

PAT を表示する (コンソール)

コンソールを使用して、のユーザ ID に関連付けられた PAT を表示できます。 CodeCatalyst

個人アクセストークンを表示するには (コンソール)

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. [個人アクセストークン] には、現在の PAT の名前と有効期限が表示されます。

PAT の表示 (CLI)

CLI を使用して、のユーザ ID に関連付けられた PAT を表示できます。 CodeCatalyst

個人アクセストークン ()AWS CLIを表示するには

- ターミナルまたはコマンドラインで、list-access-tokens以下のようにコマンドを実行します。

  ```
  aws codecatalyst list-access-tokens
  ```

  成功すると、以下の例のように、ユーザーアカウントに関連付けられた PAT に関する情報が返されます。

  ```
  {
      "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"
        }
    ]
}
```


PAT を削除する

でユーザ ID に関連付けられた PAT を削除できます。 CodeCatalyst

PAT の削除 (コンソール)

コンソールを使用して PAT を削除できます。 CodeCatalyst

個人アクセストークンを削除するには (コンソール)

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. 「個人アクセストークン」で、削除したい PAT の横にあるセレクターを選択し、「削除」を選択します。

   <name>「PAT の削除:?」 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] をクリックします。

パットの削除 (CLI)

ユーザ ID に関連付けられた PAT は、を使用して削除できます。 AWS CLIそのためには、PAT の ID を指定する必要があります。このIDは、コマンドを使用して表示できます。delete-access-token

> **Note**
>
> AWS CLI CodeCatalyst最新バージョンのを使用して作業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コマンドが含まれていない可能性があります。 AWS CLI with の使い方の詳細については 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、削除する PAT の ID delete-access-token を指定してコマンドを実行します。*#####ID # 123EXAMPLE # PAT ####################*

  ```
  aws codecatalyst delete-access-token --id a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb
  ```

  成功した場合、このコマンドは応答を返しません。

Amazon での多要素認証 (MFA) CodeCatalyst

AWS Builder ID プロファイルを個人使用または業務用に作成したかどうかにかかわらず、セキュリティの別のレイヤーとして多要素認証 (MFA) を設定することをお勧めします。スペースのメンバーで、他のユーザーとプロジェクトでコラボレーションする場合は、MFA を設定することを特におすすめします。1 つのプロジェクトには複数のユーザーがアクセスできるので、セキュリティ侵害が発生する可能性が高くなります。

MFA を有効にするときは、E メールとパスワードを使用して Amazon CodeCatalyst にサインインする必要があります。ログインのこの部分が、使い慣れた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。これが二つ目の要素で、皆さんが持っているものです。2 つ目の要素は、モバイルデバイスで生成される認証コードか、コンピューターに接続されている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。これらの複数の要素を組み合わせると、不正アクセスを防止することでセキュリティが強化されます。

多要素認証用のデバイスを登録する方法

[マイプロフィール] > [多要素認証] の次の手順に従って、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すすめします。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/ 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プロフィール」ページが開きます。
3. プロフィールページで、「プロフィー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、「デバイスを登録」を選択します。
6. 「MFA デバイスの登録」ページで、次の MFA デバイスタイプのいずれかを選択し、指示に従います。

   - 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**
        >
        > この方法はオペレーティングシステムとブラウザーによって異なるため、ブラウザー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 用に 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に時間がかかりすぎた場合、MFA デバイスは AWS Builder

ID プロファイルに正常に関連付けられていますが、MFA デバイスは同期されていません。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。これで MFA デバイスはワンタイムパスワードの生成を開始でき、使用可能になりました。

認証アプリケーション

認証アプリは、ワンタイムパスワード (OTP) ベースのサードパーティ認証アプリです。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どれも IAM Identity Center MFA で動作しますが、次の表は、選択できるよく知られたサードパーティの認証アプリの一覧です。

オペレーティングシステム	テスト済みの認証アプリ
Android	オーシー、デュオモバイル、オーセンティケーター、Microsoft LastPass オーセンティケーター、グーグルオーセンティケーター
iOS	オーシー、デュオモバイル、オーセンティケーター、Microsoft LastPass オーセンティケーター、グーグルオーセンティケーター

MFA デバイスの変更

MFA デバイスを登録したら、名前を変更したり削除したりできます。セキュリティを強化するために、常に少なくとも 1 つの MFA デバイスを有効にしておくことをおすすめします。最大 5 台のデバイスを登録できます。さらに追加する方法については、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 デバイスの名前を変更する

MFA デバイスの名前を変更するには

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プロフィール」ページが開きます。
3. プロフィールページで、「プロフィー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。ページに移動すると、[名前の変更] がグレー表示になっているのがわかります。
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。すると、モーダルがポップアップします。
6. 表示されるプロンプトで、[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が [多要素認証デバイス (MFA)] 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/ 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プロフィール」ページが開きます。
3. プロフィールページで、「プロフィー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。ページに移動すると、[削除] がグレー表示になっているのがわかります。
5. 変更する MFA デバイスを選択します。[削除] をクリックします。「MFA デバイスを削除しますか?」というモーダルが表示されます。 。指示に従ってデバイスを削除します。

6. [削除] をクリックします。削除したデバイスは、多要素認証デバイス (MFA) に表示されなくなります。

Amazon ンのセキュリティ CodeCatalyst

AWS クラウドセキュリティは最優先事項です。 AWS 顧客にとっては、セキュリティが最も重視される分野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の恩恵を受けることができます。

セキュリティは、 AWS お客様とお客様と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 AWS AWS クラウドクラウド内で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があります。 AWS また、安全に使用できるサービスも提供します。第三者監査人は、AWS、当社のセキュリティの有効性を定期的にテストおよび検証しています。に適用されるコンプライアンスプログラムについては CodeCatalyst、「AWS コンプライアンスプログラム別の対象サービス」を参照してください。
- クラウドにおけるセキュリティ — お客様の責任は、 AWS 使用する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 を使用する際に責任分担モデルを適用する方法を理解するのに役立ちます CodeCatalyst。 CodeCatalyst セキュリティとコンプライアンスの目標を満たすように設定する方法を示しています。また、 AWS CodeCatalyst リソースの監視と保護に役立つ他のサービスの使い方についても学びます。

コンテンツ

- CodeCatalyst トラストモデル

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、「Amazon での多要素認証 (MFA) CodeCatalyst」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、「Amazon での多要素認証 (MFA) CodeCatalyst」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口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、 Builder ID AWS を作成して使用し、サインインしてスペースとプロジェクトにアクセスします。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AWS アカウントしてその でリソースを作成および使用する場合、 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するために役立つ AWS のサービスです。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、無料で使用できる AWS のサービスです。

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AWS アカウントとして を接続する必要があります。 CodeCatalyst スペースを検証するAWS アカウントには、 に管理者権限を持っているか、 アクセス許可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

の作成とアクセス CodeCatalyst に使用できる IAM ロールをスペースに追加するオプションもあります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 で請求されます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 で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
- AWS リソースへの Amazon CodeCatalyst アクセス用の IAM ロール

IAM で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、および条件を制御します。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、それがアタッチされてい

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学ぶには、『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 に対してアクションを実行できるかということです。

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、関連する 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 に対してアクションを実行できるかということです。

Resource JSON ポリシーの要素は、オブジェクトあるい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は、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、どの条件下でどのアクションを実行できるかということです。

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。equal や less than などの条件演算子を使用して条件式を作成することによって、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 要素が指定されている場合、または 1 つの Condition 要素に複数のキーが指定されている場合、AWS では 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が指定されている場合、AWS では 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ています。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には 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、 ユーザーおよびロールには、 CodeCatalyst リソースを作成または変更する権限はありません。また、AWS Management Console、AWS Command Line Interface (AWS CLI)、または AWS API を使用してタスクを実行することもできません。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: 単一の で接続リクエストを受け入れることをユーザーに許可する 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 と 間の接続のリクエストのみを表示 CodeCatalyst および受け入れることができますAWS アカウント。さらに、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 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: コンソールで単一の 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。このポリシーでは、 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 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、で 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、 CodeCatalyst と 間の接続を管理する機能をユーザーに拒否します 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、接続アカウント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。
AssociateIamRoleTo Connection	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	arn:aws:codecataly st:region: *account_I*

CodeCatalyst アカウント接続の アクセス許可	必要な許可	リソース
	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。	*D* :/connect ions/ *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:codecataly st:region: *account_I D* :/connect ions/ *connection_ID*
DisassociateIamRol eFromConnection	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:codecataly st:region: *account_I D* :/connect ions/ *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:codecataly st:region: *account_I D* :/connect ions/ *connection_ID*
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:codecataly st:region: *account_I D* :/connect ions/ *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
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	ID をスペースの CodeCatalyst IAM Identity Center アプリケーション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*
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 アプリケーションを作成するのに必要です。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*
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*
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 アプリケーションの CodeCatalyst スペースと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
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 allIAM 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。	`arn:aws:codecatalyst:region: account_ID :/identity-center-applications/ identity-center-application_ID`
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*
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
AmazonCodeCatalyst ReadOnlyAccess - 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

AWS リソースへの Amazon CodeCatalyst アクセス用の IAM ロール

CodeCatalyst は、 AWS アカウントを CodeCatalyst スペースに接続することで 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 ポリシー要素のリファレンス」を参照してください。

- CodeCatalyst プロジェクトおよびワークフローAWS アカウントの のリソースにアクセスするには、まず にそれらのリソース CodeCatalyst へ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、接続された でサービスロールを作成し、スペース内のユーザーとプロジェクトに代わって AWS アカウント が CodeCatalyst引き受けることができるように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、これらのロールには必要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 トラストモデル。

- 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に 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、スペースの作成時にサービスロールを追加し、そのロールCodeCatalystWorkflowDevelopmentRole-*spaceName*の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、スペース内のプロジェクト内のユーザーに代わって がロールを引き受け 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用の 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**
>
> このロールは開発用アカウントでのみ使用が推奨され、 AdministratorAccessAWSマネージド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 トラストモデル。

```
"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用の CodeCatalystWorkflowDevelopmentRole-*spaceName* ロールの作成

次のステップに従って、スペース内のワークフローに使用するCodeCatalystWorkflowDevelopmentRole-*spaceName*ロールを作成します。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。で AWS アカウントおよび IAM ロール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 アカウント スペースの管理。

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、スペース AWS アカウントに対して同じ 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. ロールを作成する AWS アカウント 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AWS Management Consoleを選択します。

   「Amazon CodeCatalyst スペースに IAM ロールを追加する」ページが で開きます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 開発管理者ロールを作成するを選択します。このオプションでは、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し、 AdministratorAccessAWSマネージド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する場合にのみ推奨されます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*ロールのリストに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

スペース内の 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、スペースの作成時にサービスロールを追加し、そのロールのAWSRoleForCodeCatalystSupportオプションを選択します。これにより、ポリシーとロールが作成されるだけでなく、スペース内のプロジェクト内のユーザーに代わって がロールを引き受け 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用の 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、アクセス許可をサポートするアクセス許可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、 がロールを引き受け CodeCatalyst 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用の AWSRoleForCodeCatalystSupport ロールの作成

次のステップに従って、スペース内のサポートケースに使用する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。で AWS アカウントおよび IAM ロール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 アカウント スペースの管理。

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、スペース AWS アカウントに対して同じ 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. ロールを作成する AWS アカウント 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AWS Management Consoleを選択します。

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、サインインが必要になる場合があります。

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。このオプションでは、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、AWSRoleForCodeCatalystSupport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にし、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`*ロールのリストに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、 アカウントで使用するために作成できる 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のロールを手動で作成する。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 アクセスの ビルドロール

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、内のAWS CloudFormationリソースでタスクを実行する CodeCatalyst ために必要な、スコープ付きの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、次の操作を実行する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。
- による 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**
>
> ロールがワークフローアクションの実行に初めて使用されるときは、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、利用可能になった後にリソース名を使用してポリシーの範囲を絞り込みます。
>
> ```
> "Resource": "*"
> ```

CodeCatalyst の ビルドロール 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、 内のAWS CloudFormationリソースでタスクを実行する CodeCatalyst ために必要な、スコープ付きの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、次の操作を実行するアクセス許可を付与します。

- による 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 管理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 ビルドロール

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、 内の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、スコープ付きの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、次の操作を実行する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**
>
> ロールがワークフローアクションの実行に初めて使用されるときは、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、利用可能になった後にリソース名を使用してポリシーの範囲を絞り込みます。
>
> ```
> "Resource": "*"
> ```

CodeCatalyst の ロールのデプロイ AWS CloudFormation

を使用する CodeCatalyst ワークフローデプロイアクションの場合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 ために必要なアクセス許可が限定されたポリシーを使用できますAWS アカウント。

このロールは、次の操作を実行するアクセス許可を付与します。

- CodeCatalyst が TAK‧ 関数を呼び出して、 を介してブルー/グリーンデプロイを実行できるようにしますAWS CloudFormation。
- CodeCatalyst が でスタックと変更セットを作成および更新できるようにします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**
>
> ロールがワークフローアクションの実行に初めて使用されるときは、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、利用可能になった後にリソース名を使用してポリシーの範囲を絞り込みます。
>
> ```
> "Resource": "*"
> ```

CodeCatalyst Amazon EC2 の ロールのデプロイ

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、スコープ付きの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、次の操作を実行する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**
>
> ロールがワークフローアクションの実行に初めて使用されるときは、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、利用可能になった後にリソース名を使用して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、スコープ付きの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、次の操作を実行する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**
>
> ロールがワークフローアクションの実行に初めて使用されるときは、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、利用可能になった後にリソース名を使用してポリシーの範囲を絞り込みます。
>
> ```
> "Resource": "*"
> ```

CodeCatalyst Lambda 用の 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、スコープ付きの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、次の操作を実行する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、更新、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 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**
>
> ロールがワークフローアクションの実行に初めて使用されるときは、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、利用可能になった後にリソース名を使用してポリシーの範囲を絞り込みます。
>
> ```
> "Resource": "*"
> ```

CodeCatalyst Lambda 用の ロールのデプロイ

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、スコープ付きの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、次の操作を実行する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、更新、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": [
```

```
                "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**
>
> ロールがワークフローアクションの実行に初めて使用されるときは、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、利用可能になった後にリソース名を使用してポリシーの範囲を絞り込みます。
>
> ```
> "Resource": "*"
> ```

CodeCatalyst の ロールのデプロイ 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、 が 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、スコープ付きの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、次の操作を実行する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 CLI AWS SAM 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 が でスタックと変更セットを作成および更新できるようにします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**
>
> ロールがワークフローアクションの実行に初めて使用されるときは、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、利用可能になった後にリソース名を使用して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、スコープ付きの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、次の操作を実行する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**
>
> ロールがワークフローアクションの実行に初めて使用されるときは、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、利用可能になった後にリソース名を使用して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、スコープ付きの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、次の操作を実行する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**
>
> ロールがワークフローアクションの実行に初めて使用されるときは、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、利用可能になった後にリソース名を使用して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、スコープ付きの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る。
- 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**
>
> ロールがワークフローアクションの実行に初めて使用されるときは、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、利用可能になった後にリソース名を使用して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のロールを手動で作成する

CodeCatalyst ワークフローアクションでは、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 IAM ロールを使用します。

IAM でこれらのロールを作成するには、次の手順に従います。

デプロイロールを作成するには

1. ロールのポリシーを次のように作成します。

   a. AWS にサインインします。
   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**
>
> ロールがワークフローアクションの実行に初めて使用されるときは、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、利用可能になった後にリソース名を使用して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 に、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. 次のようにデプロイ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-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. [ロールの作成] 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-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. 上部に 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が作成され、その ARN が取得されました。

ビルドロールを作成するには

1. ロールのポリシーを次のように作成します。

a. AWS にサインインします。

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**
>
> ロールがワークフローアクションの実行に初めて使用されるときは、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、利用可能になった後にリソース名を使用して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 に、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. ビルド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-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. [ロールの作成] 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含む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、先ほど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. 上部に 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が作成され、その ARN が取得されました。

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、この手順でさらに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、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. [次へ: 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工ラー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. [ロールの作成] 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、先ほど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し、 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設定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短縮し、 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を手動で作成する

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 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. ロールのポリシーを次のように作成します。

    a. AWS にサインインします。

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**
>
> ロールがワークフローアクションの実行に初めて使用されるときは、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、利用可能になった後にリソース名を使用して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 に、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. ビルド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-webapp-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-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. [ロールの作成] 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含む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がビルドロールにアタッチされました。ビルドロールには、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、先ほど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. 上部に 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が作成され、その ARN が取得されました。

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 CloudFormationと SAM のデプロイロールと呼ばれる 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. ロールのポリシーを次のように作成します。

   a. AWS にサインインします。

   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**
>
> ロールがワークフローアクションの実行に初めて使用されるときは、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、利用可能になった後にリソース名を使用して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 に、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. ビルド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. [ロールの作成] 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含む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がビルドロールにアタッチされました。ビルドロールには、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、先ほど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. 上部に 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が作成され、その ARN が取得されました。

SAM のデプロイロールを作成するには

1. ロールのポリシーを次のように作成します。

a. AWS にサインインします。

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**
>
> ロールがワークフローアクションの実行に初めて使用されるときは、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、利用可能になった後にリソース名を使用して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 に、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. ビルド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. [ロールの作成] 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含む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. 選択codecatalyst-SAM-deploy-roleすると、詳細が表示されます。

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がビルドロールにアタッチされました。ビルドロールには、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、先ほど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. 上部に 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が作成され、その ARN が取得されました。

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対象で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による対象範囲内の AWS のサービス のサービス」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。

AWS Artifact 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。詳細については、「AWS Artifact におけるダウンロードレポート」「」を参照してください。

AWS のサービス を使用する際のユーザーのコンプライアンス責任は、ユーザーのデータの機密性や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。AWS では、コンプライアンスに役立つ次のリソースを提供しています。

- 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を AWS にデプロイするための手順を示しています。
- 「Amazon Web Services での HIPAA のセキュリティとコンプライアンスのためのアーキテクチャ」- このホワイトペーパーは、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を説明しています。

  > **Note**
  >
  > すべての AWS のサービス が HIPAA 適格であ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 コンプライアンスのリソース – このワークブックおよびガイドのコレクションは、顧客の業界と拠点に適用されるものである場合があります。
- 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分担モデルを理解してください。このガイドでは、AWS のサービスセキュリティを確保するためのベストプラクティスをまとめ、複数のフレームワーク (米国標準技術研究所 (NIST)、ペイメントカード業界セキュリティ標準評議会 (PCI)、国際標準化機構 (ISO) など) にわたるセキュリティ管理にガイダンスをまとめています。
- AWS Config デベロッパーガイド のルールでのリソースの評価 - AWS Config サービスでは、自社のプラクティス、業界ガイドライン、および規制に対するリソースの設定の準拠状態を評価します。
- AWS Security Hub - この AWS のサービス は、AWS 内のセキュリティ状態の包括的なビューを提供します。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。
- AWS Audit Manager - この AWS のサービス は AWS の使用状況を継続的に監査し、リスクの管理方法やコンプライアンスを業界スタンダードへの準拠を簡素化するために役立ちます。

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 トラストモデル

Amazon CodeCatalyst ントラストモデルでは CodeCatalyst 、AWS アカウント接続時にサービスロールを引き受けることができます。このモデルは IAM ロール、 CodeCatalyst サービスプリンシパル、スペースを接続します。 CodeCatalyst 信頼ポリシーは、`aws:SourceArn`条件キーを使用して、 CodeCatalyst 条件キーで指定されたスペースにアクセス権限を付与します。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権限が付与され、Conditionその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ています。

```
"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は、 CodeCatalyst スペース ID の Amazon リソースネーム (ARN) aws:SourceArn を含む条件キーを通じてアクセス権を付与されます。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、aws:SourceArnなどの条件キーでのみ使用してください。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、ポリシー内の権限の範囲をできるだけ小さくしてください。

- aws:SourceArn条件キーにワイルドカード (*) を使用すると、のスペース内のすべてのプロジェクトを指定できます。project/*
- のスペース内の特定のプロジェクトについて、aws:SourceArn条件キーにリソースレベルの権限を指定できます。project/*projectId*

のサービスプリンシパル CodeCatalyst

リソースベースの JSON Principal ポリシーの要素を使用して、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。ID Principal ベースのポリシーではこの要素を使用できません。同様に、ユーザーグループをポリシー (リソースベースのポリシーなど) のプリンシパルとして識別することはできません。なぜなら、グループは認証ではなくアクセス権限に関係し、プリンシパルは認証された IAM エンティティだからです。

信頼ポリシーでは、AWS のサービスPrincipalリソースベースのポリシーの要素またはプリンシパルをサポートする条件キーで指定できます。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。以下のサービスプリンシパルが定義されています。 CodeCatalyst

- codecatalyst.amazonaws.com-このサービスプリンシパルは、へのアクセスを許可するロールに使用されます。 CodeCatalyst AWS
- codecatalyst-runner.amazonaws.com-このサービスプリンシパルは、ワークフローのデプロイメント内のリソースへのアクセスを許可するロールに使用されます。 CodeCatalyst AWS CodeCatalyst

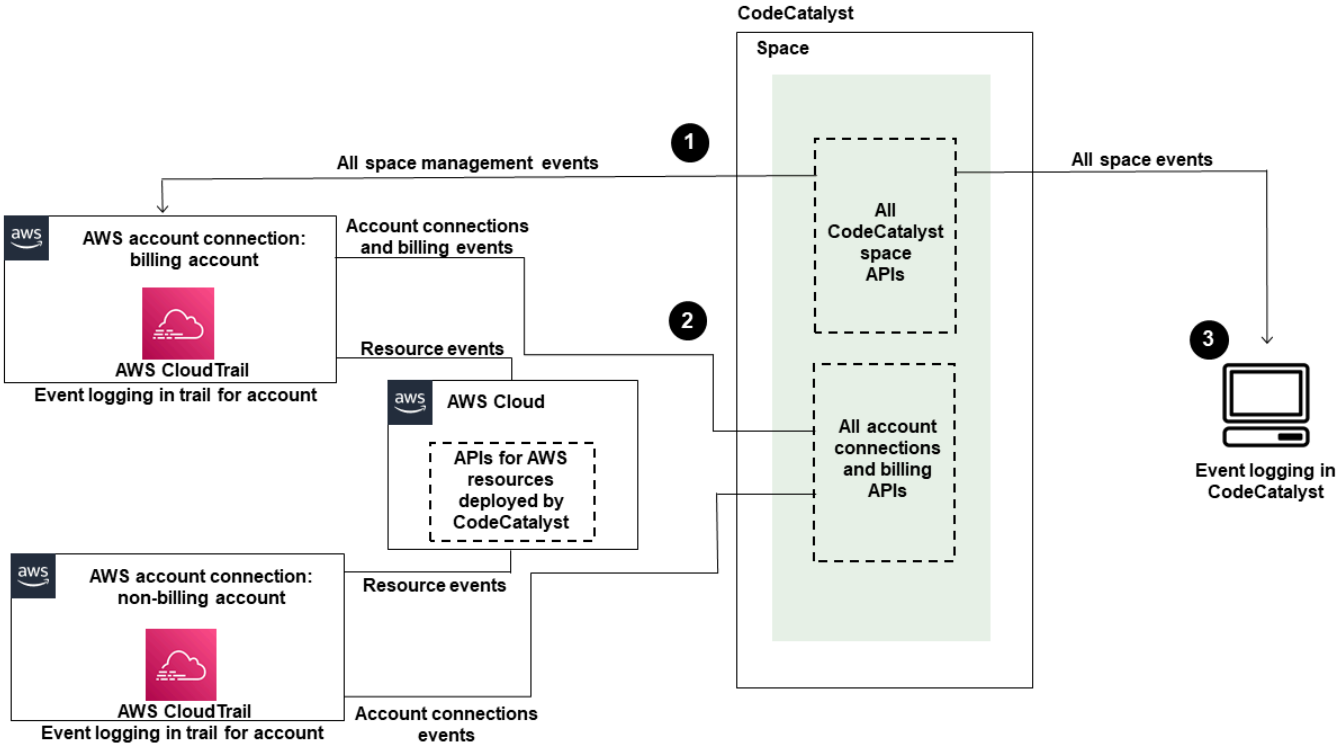
詳細については、『IAM ユーザーガイド』の「JSON ポリシー要素:プリンシパル」を参照してください。AWS

Amazon でのモニタリング CodeCatalyst

Amazon では CodeCatalyst、 AWS CloudTrail スペースの管理イベントはスペースの請求アカウントの記録によって収集され、記録されます。 CloudTrail CodeCatalyst 口ギングはイベントの口ギングを管理する主な方法で、もう 1 つはイベント口グインを確認する方法です CodeCatalyst。

アカウント内のイベントは、に設定されたトレイルと指定されたバケットとともに記録されます 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にログインされるのに対し、 AWS CloudTrail アカウント接続/請求イベントとリソースイベントはそれぞれのアカウントにログインされる様子を示しています。 CloudTrail



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、が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。 AWS アカウント CloudTrail 使用される証跡は請求先アカウント用に作成された証跡で、スペースイベントが記録されます。 CloudTrail スペースによって、 CodeCatalyst またはスペースに代わって行われた API 呼び出しと関連イベントをキャプチャし、指定した S3 バケットにログファイルを配信します。請求先アカウントが別の AWS アカウントに変更された場合、スペースイベントはそのアカウントのトレイルとバケットに記録されます。 CodeCatalyst によって記録される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 内の情報 CloudTrail。
2. 請求先アカウントなど、スペースに接続されている他のアカウントは、アカウント接続と請求イベントのサブセットを記録します。 CodeCatalyst AWS そのアカウントにデプロイされたリソースのアカウントイベントを生成するワークフローも、のトレイルとバケットに記録されます。 AWS アカウント CloudTrail スペースによって、 CodeCatalystまたはスペースに代わって行われた API 呼び出しと関連イベントをキャプチャし、指定した S3 バケットにログファイルを配信します。 CodeCatalyst によって記録される管理イベントの詳細については CloudTrail、を参照してくださいログに記録されたイベントへのアクセス CodeCatalyst。

3. list-event-logsコマンドを使用して、 CodeCatalyst スペース内の特定の時間内のスペース内のアクションを監視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。 CodeCatalyst スペース内のアクションのイベント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。詳細については、「ログに記録されたイベントへのアクセス CodeCatalyst」を参照してください。

> **Note**
>
> `ListEventLogs`特定のスペースでの過去 30 日間のイベントを保証します。また、過去 90 CodeCatalyst AWS CloudTrail 日間の管理イベントのリストをコンソールで表示したり取得したりするには、イベント履歴を表示するか、過去 90 日間にわたるイベントの記録を作成して管理するための証跡を作成します。詳細については、「 CloudTrail イベント履歴の操作」と「記録の操作」を参照してください CloudTrail 。

> **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、CloudTrail CodeCatalyst スペースのロギングの一部として記録されません。たとえば、CodeCatalyst リソースにはスペースやプロジェクトが含まれます。 AWS リソースには Amazon ECS サービスまたは Lambda 関数が含まれます。 CloudTrail AWS アカウント リソースがデプロイされる場所ごとに個別にロギングを設定する必要があります。

では、イベント監視のフローとして考えられるものを 1 つ紹介します CodeCatalyst。

Mary Major CodeCatalyst はスペースのスペース管理者で、 CodeCatalyst ログインし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のすべての管理イベントを表示します。 CloudTrailCodeCatalyst 内の情報 CloudTrailロギン中のイベントの例を参照してください。 CloudTrail

Dev Environments CodeCatalyst などので作成されたリソースについては、Mary はスペースの請求先アカウントのイベント履歴を確認し、でプロジェクトメンバーが Dev Environments を作成したイベントを調査します。 CodeCatalystこのイベントは、開発環境を作成したユーザーの ID ストア IAM ID AWS タイプとビルダー ID の認証情報を提供します。のワークフローによってデプロイされるリソース (サーバーレスデプロイ用の Lambda 関数など) AWS 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 (接続先アカウントでもある CodeCatalyst) トレイルのイベント履歴を表示できます。 CodeCatalyst

さらに詳しく調べるために、Mary list-event-logsは内のコマンドを使用してスペース内のすべての CodeCatalyst API のイベントを表示することもできます。 AWS CLI

トピック

- CodeCatalyst AWS アカウント接続中の API 呼び出しのロギング AWS CloudTrail
- ログに記録されたイベントへのアクセス CodeCatalyst

CodeCatalyst AWS アカウント接続中の API 呼び出しのロギング AWS CloudTrail

Amazon CodeCatalyst はAWS CloudTrail、ユーザ、ロール、AWS のサービスまたはによって実行されたアクションの記録を提供するサービスと統合されています。 CloudTrail 接続されているユーザーに代わって行われた API CodeCatalyst AWS アカウント 呼び出しをイベントとしてキャプチャします。証跡を作成すると、 CloudTrail のイベントを含むイベントを S3 バケットに継続的に配信できるようになります。 CodeCatalyst証跡を設定しなくても、 CloudTrail コンソールの [イベント履歴] で最新のイベントを確認できます。

CodeCatalyst CloudTrail 次のアクションをイベントとしてログファイルに記録することをサポートします。

- CodeCatalyst スペースの管理イベントは、AWS アカウントそのスペースに指定されている請求アカウントに記録されます。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ログに記録されたイベントへのアクセス CodeCatalyst。

- CodeCatalyst 接続された環境で発生する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、AWS アカウント接続先のイベントとして記録されます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**
>
> 1 つのスペースには複数のアカウントを関連付けることができますが、 CloudTrail CodeCatalyst スペースやプロジェクト内のイベントの記録は請求先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 CodeCatalyst AWS無料利用枠を超えたリソースに対して課金されるアカウントです。1 つのスペースには複数のアカウントを接続できますが、請求アカウントに指定できるのは 1 つのアカウントだけです。請求アカウントまたはスペースの追加の接続アカウントには、Amazon ECS クラスターや S3 AWS バケットなどのリソースとインフラストラクチャーをワークフローからデプロイするために使用される IAM ロールを設定できます。CodeCatalyst ワークフロー YAML を使用して、デプロイ先を識別できます。AWS アカウント

> **Note**
>
> AWS CodeCatalyst ワークフローの接続アカウントにデプロイされたリソースは、CloudTrail CodeCatalyst スペースのロギングの一部として記録されません。たとえば、CodeCatalystリソースにはスペースやプロジェクトが含まれます。 AWSリソースには Amazon ECS サービスまたは Lambda 関数が含まれます。 CloudTrail ロギングは、AWS アカウントリソースがデプロイされる場所ごとに個別に設定する必要があります。

CodeCatalyst 接続アカウントへのログインには以下の考慮事項があります。

- CloudTrail イベントへのアクセスは、接続されているアカウントではなく、接続されているアカウントの IAM CodeCatalyst で管理されます。
- GitHub リポジトリへのリンクなどのサードパーティ接続では、 CloudTrail サードパーティのリソース名がログに記録されます。

> **Note**
>
> CloudTrail CodeCatalyst イベントのロギングはスペースレベルで行われ、プロジェクトの境界によってイベントが分離されるわけではありません。

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、 CloudTrail CodeCatalystログインしているスペースとに接続されているすべてのイベントのロギングについて説明します CodeCatalyst。AWS アカウントさらに、 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 CLIaws codecatalyst list-event-logsおよびコマンドを使用することもできます。詳細については、「ログに記録されたイベントへのアクセス CodeCatalyst」を参照してください。

CodeCatalyst スペースイベント

CodeCatalyst スペースレベルとプロジェクトレベルのリソースを管理するためのアクションは、スペースの請求アカウントに記録されます。 CloudTrail CodeCatalyst スペースのロギングでは、以下の考慮事項に基づいてイベントが記録されます。

- CloudTrail イベントはスペース全体に適用され、特定のプロジェクトを対象とするものではありません。
- CodeCatalyst スペースに接続すると、アカウント接続に関するログ可能なイベントがそのスペースに記録されます。AWS アカウント 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。
- スペースに接続し、 CodeCatalyst そのスペースの請求先アカウントとして指定すると、AWS アカウントイベントがそのスペースに記録されます。AWS アカウントこの接続をいったん有効にすると、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、請求先アカウントにのみ記録されます。 CloudTrail 宛先アカウントを変更するには、で請求先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、 CodeCatalyst変更内容が新しい請求先アカウントに反映されます。その後、 CloudTrail移行先アカウントが更新されます。

AWS CodeCatalystスペースレベルとプロジェクトレベルのリソースを管理するためのアクションに関連するイベントの例を以下に示します。次の API は SDK と CLI を通じてリリースされています。イベントは、AWS アカウント CodeCatalyst スペースの請求アカウントとして指定された場所に記録されます。

- `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

アカウント接続や請求に関するアクションに関連するイベントの例を以下に示します。AWS CodeCatalyst

- AcceptConnection
- AssociateIAMRoletoConnection
- DeleteConnection
- DissassociateIAMRolefromConnection
- GetBillingAuthorization
- GetConnection
- GetPendingConnection
- ListConnections
- ListIAMRolesforConnection
- PutBillingAuthorization
- RejectConnection

CodeCatalyst 内の情報 CloudTrail

CloudTrail AWS アカウントアカウントを作成すると、で有効になります。これをスペースに接続すると、AWS アカウント CodeCatalyst AWS アカウントそのスペースで発生したそのスペースのイベントは、その AWS アカウントにログインされます。 CloudTrail CodeCatalyst 記録可能なイベントは、 CloudTrail そのアカウントの他の記録可能なイベントとともに、 CloudTrail CloudTrail AWS接続されたアカウントのログとコンソールのイベント履歴にイベントとして記録されます。

各イベントまたはログエントリには、リクエストの生成者に関する情報が含まれます。ID 情報は次の判断に役立ちます。

- リクエストがビルダー ID を持つユーザーによって行われたかどうか。AWS
- リクエストが、ルート認証情報と AWS Identity and Access Management (IAM) ユーザー認証情報のどちらを使用して送信されたか。
- リクエストがロールまたはフェデレーティッド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 のサービスによって送信されたかどうか。

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

CodeCatalystでのアクティビティに関するイベントを含めAWS アカウント、社内のイベントを継続的に記録するにはAWS アカウント、トレイルを作成してください。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 に適用されますAWS リージョン 証跡では、AWS パーティションのすべてのリージョンからのイベントがログに記録され、指定した S3 バケットにログファイルが配信されます。さらに、AWS CloudTrail ログに収集されたイベントデータをさらに分析して処理するように他のサービスを設定することもできます。詳細については、次を参照してください。

- 追跡を作成するための概要
- CloudTrail サポート対象のサービスとインテグレーション
- CloudTrail の Amazon SNS 通知の設定
- CloudTrail 複数の地域からのログファイルの受信、 CloudTrail 複数のアカウントからのログファイルの受信

証跡とは、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。イベントはあらゆる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に関する情報、アクションの日時、リクエストパラメータなどが含まれます。 CloudTrail ログファイルはパブリック API 呼び出しの順序付けられたスタックトレースではないため、特定の順序で表示されることはありません。

CodeCatalyst のアカウント接続イベントの例 AWS

次の例は、 CloudTrail `ListConnections`アクションを示すログエントリを示しています。AWS アカウントスペースに接続されているものについては、`ListConnections`そのスペース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。 CodeCatalyst AWS アカウントAWS アカウントイベントは指定された in に記録され`accountId`、の値はアクションに使用されたロールの 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"
}
```

CodeCatalyst のプロジェクトリソースイベントの例 AWS

次の例は、 CloudTrail CreateDevEnvironmentアクションを示すログエントリを示しています。スペースに接続され、スペースの指定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。AWS アカウント

accountIdフィールドではuserIdentity、これがすべての AWS Builder ID ID の ID プールをホストする IAM ID センターのアカウント ID (432677196278) です。このアカウント ID には、CodeCatalyst イベントのユーザーに関する以下の情報が含まれています。

- typeこのフィールドには、リクエストの IAM エンティティの種類が表示されます。CodeCatalyst スペースリソースとプロジェクトリソースのイベントの場合、この値はですIdentityCenterUser。accountIdこのフィールドには、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このフィールドには、AWSユーザーのビルダー 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このフィールドには、スペースの請求先アカウントのアカウント ID が含まれ、例の値は 111122223333 です。

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**
>
> イベントによっては、ユーザーエージェントが不明な場合があります。この場合、CodeCatalyst `UnknownuserAgent` CloudTrail はイベントのフィールドにの値を入力します。

CodeCatalyst イベントトレイルのクエリ

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。

ログに記録されたイベントへのアクセス CodeCatalyst

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。 AWS CLI を使用して、指定した期間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した 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**
>
> CodeCatalyst スペースの管理イベントは、 CloudTrail 接続された請求アカウントに記録されます。 CodeCatalyst によって記録される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 内の情報 CloudTrail。

スペースのイベントログを表示するには、をインストールしてプロファイルを設定し CodeCatalyst、 AWS CLI その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**
>
> connected に代わって発生するイベントのログを表示したり AWS アカウント、接続されている請求先アカウントのスペースやプロジェクトリソースのイベントのログを表示したりするには、を使用できます AWS CloudTrail。 CodeCatalyst 詳細については、「 CodeCatalyst AWS アカウント接続中の API 呼び出しのロギング AWS CloudTrail」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以下を指定してコマンドを実行します。

   - **--space-name**オプション付きのスペースの名前。
   - イベントのレビューを開始する日付と時刻。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 (オプション)。**--start-time**
   - イベントのレビューを停止したい日付と時刻。RFC 3339 で指定されている世界協定時刻 (UTC) タイムスタンプ形式 (オプション)。 **--end-time**
   - (オプション) 1 回の応答で返される結果の最大数 (オプション付き)。**--max-results**結果の数が指定した数よりも多い場合、nextToken応答には次の結果を返すために使用できる要素が含まれます。
   - (オプション) オプションを使用して、返したい特定のイベントタイプに結果を制限します。**--event-name**

   この例では、*2022-11-30 ExampleCorp## 2022-12-01* までの期間の口グに記録されたイベントが返され、応答では最大 *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. この期間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. list-event-logs--next-tokenオプションと返されたトークンの値を指定してコマンドを再実行し、リクエストと一致するログに記録された次のイベントセットを取得します。

の ID、許可、アクセスのクォータ CodeCatalyst

次の表は、Amazon におけるアイデンティティ、許可、アクセスのクォータと制限について説明しています。 CodeCatalystAmazon 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	長さが 3 ~ 100 文字の任意の組み合わせで、先頭は文字でなければなりません。有効な文字:A ~ Z、a ~ z、および 0-9。エイリアスには以下のことはできません。 • 3 文字未満で指定してください。 • スペースまたは以下の文字を含める: ? ^ * [\ ~ :
1 人のユーザーが 1 日に送信する招待状の最大数	500
1 つのメールアドレスに送信される招待状の 1 日あたりの最大数	25
ユーザー 1 人あたりのパーソナルアクセストークン (PAT) の最大数	100
のパスワード CodeCatalyst	8 文字から 64 文字までの任意の文字の組み合わせ。有効な文字:A ~ Z、a ~ z、および 0-9。パスワードには以下の英数字以外の文字を使用できます。(~ ! @ # $ % ^ & * _ - + = ` | \ { } [ ] : ; " ' < > , . ? /)
の PAT 名 CodeCatalyst	1 文字から 100 文字までの任意の組み合わせを使用できます。
プロジェクトメンバーへの招待の有効期限が切れるまでの時間	24 時間後に有効期限が切れます。

リソース	情報
スペースメンバー招待の有効期限が切れるまでの時間	24 時間後に有効期限が切れます。
E メールアドレスの検証の有効期限が切れるまでの時間	送信後 10 分で有効期限が切れます。

トラブルシューティング

このセクションでは、Amazon CodeCatalyst プロフィールにアクセス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。

サインアップに関する問題

サインアップ中に問題が発生する可能性があります。いくつかの解決策があります。

私のメールアドレスはすでに使用されています。

入力したメールアドレスがすでに使用されていて、自分のものだとわかっている場合は、すでにプロフィールを作成している可能性があります。この既存の ID でサインインしてください。既存のメールアドレスを所有していない場合は、別の未使用のメールアドレスでサインアップしてください。

メールの確認を完了させることができない

確認メールが届いていない場合

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てください。迷惑メール、迷惑メール、削除済みアイテムフォルダをもう一度確認してください。

3. それでも確認メールが表示されない場合は、[コードを再送信] を選択します。そのページをすでに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開してください。

私のパスワードは最低要件を満たしていません。

セキュリティ上の理由から、パスワードには 8 ~ 20 文字、大文字、小文字、数字を含める必要があります。

サインインの問題

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、必ず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。
- パスワードは 8 ~ 64 文字の長さで、大文字と小文字の両方、数字、および英数字以外の文字を少なくとも 1 文字含む必要があります。
- 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、Amazon での多要素認証 (MFA) CodeCatalyst 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は、登録できる MFA デバイスの上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。

サインアウトに問題があります。

サインアウトする場所が見つからない

ページの右上隅にある [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。

ワークフローが失敗すると、「ロールが存在しません」というエラーが表示されます。

問題:Web アプリケーションまたはサーバレスブループリントからプロジェクトを作成すると、ワークフローが次のエラーで失敗します。

CLIENT_ERROR: ロールが存在しません

考えられる解決策:ワークフローを実行する権限を持つ IAM ロールを設定し、その IAM ロールをワークフロー YAML に追加しても、IAM ロールをアカウント接続に追加する必要がある場合があるため、ワークフローは引き続き失敗します。で説明されているように、IAM ロールをスペースの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

失敗したワークフローのロールエラーが出ます。

問題:Web アプリケーションまたはサーバレスブループリントからプロジェクトを作成すると、ワークフローが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、存在しません

考えられる解決策:プロジェクトが作成されたスペースでは、接続の設定が必要な場合や、 AWS アカウント 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。 AWS アカウント スペースにすでにアクティブな接続がある場合は、ワークフローアクションを実行する権限を持つ 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

考えられる解決策:接続を指定せずにプロジェクトを作成した場合、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。 AWS アカウント スペースにすでにアクティブな接続と 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 ロールを含む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。デプロイ環境へのアカウント接続と IAM ロールの追加

プロジェクトワークフローの IAM ロールを更新する必要があります。

AWS アカウント 接続の設定が完了し、IAM ロールが作成されてアカウント接続に追加されたら、プロジェクトワークフローの 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 を更新した 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の場合、ワークフローは自動的に開始されます。それ以外の場合、ワークフローを再実行するには [実行] を選択します。

サポートフォームにはどのように記入すればよいですか?

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。

の拡張機能 CodeCatalyst

Amazon CodeCatalyst には、機能の追加や外部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うと、チームはエクスペリエンスをカスタマイズできます CodeCatalyst。

トピック

拡張機能の概念

ここでは、で拡張機能を扱う際に知っておく必要がある概念と用語をいくつか紹介します CodeCatalyst。

拡張子

エクステンションは、 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たり、外部のサービスと統合したりできるアドオンです CodeCatalyst。エクステンションはカタログからブラウズしてインストールできます。 CodeCatalyst

CodeCatalyst カタログ

CodeCatalyst カタログは、 CodeCatalystで使用可能なすべての拡張機能を一元的に一覧にしたものです。 CodeCatalyst カタログを閲覧して、ソース、 CodeCatalyst ワークフローなどの分野でチームのエクスペリエンスを向上させることができる拡張機能を見つけることができます。

での拡張機能のインストールとアンインストール CodeCatalyst

エクステンションは、 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たり、外部のサービスと統合したりできるアドオンです。 CodeCatalyst拡張機能はカタログから参照してインストールできます。 CodeCatalyst

トピック

拡張機能のインストール

CodeCatalyst スペース内のプロジェクトに機能を追加するエクステンションをスペースにインストールできます。 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。

> **Important**
>
> 拡張機能をインストールするには、スペースの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。

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/ 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。
3. CodeCatalyst

   トップメニューのカタログアイコンを選択してカタログに移動します。エクステンションを検索したり、カテゴリーに基づいてエクステンションを絞り込んだりできます。
4. (オプション) エクステンションの名前を選択すると、エクステンションに付与される権限など、エクステンションに関する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、インストールした拡張機能の詳細ページが表示されます。タブを参照して、拡張機能に関する詳細情報を確認してください。詳細ページでは、必要に応じて拡張機能の詳細設定を行うこともできます。

拡張機能のアンインストール

CodeCatalyst 以前にスペースにインストールされていた拡張機能を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、 CodeCatalyst その拡張機能に関連するリソースがスペースまたはプロジェクトから削除される場合があります。

> **Important**
>
> 拡張機能をアンインストールするには、スペースの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。

CodeCatalyst スペースから拡張機能をアンインストールするには

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。
3. 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するには、次のいずれかを実行します。

   a. [設定] を選択し、[インストール済みの拡張機能] を選択します。

   b. 

      プメニューのカタログアイコンを選択します。

トッ
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。指示に従い、[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 でのリポジトリの使用 CodeCatalyst

GitHub は、開発者がコードを保存および管理するのに役立つ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 GitHub CodeCatalyst プロジェクトでリンクされたリポジトリを使用できます。新しい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。 CodeCatalyst 詳細については、「GitHubリポジトリをリンクしてプロジェクトを作成する」を参照してください。

> **Note**
>
> GitHub 空のリポジトリやアーカイブされたリポジトリはプロジェクトには使用できません。CodeCatalyst GitHub リポジトリエクステンションは Enterprise Server GitHub 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、次のことができるようになります。

- GitHub にあるソースリポジトリのリストにリポジトリが表示されます。 CodeCatalyst
- ワークフロー定義ファイルをリポジトリに保存して管理します。 GitHub
- GitHub リンクされたリポジトリに保存されているファイルを Dev Environments CodeCatalyst から作成、読み取り、更新、削除する
- Start CodeCatalyst ワークフローは、コードがリポジトリにプッシュされると自動的に実行されます。 GitHub
- GitHub リンクされたリポジトリのソースファイルをワークフローで使用します。 CodeCatalyst
- GitHub CodeCatalyst ワークフロー内のアクションを読み取って実行する

以下のトピックでは、GitHub リポジトリ拡張機能をインストールして設定する方法、 GitHub およびリポジトリをプロジェクトにリンクした後の使用方法について説明します。 CodeCatalyst

トピック

クイックスタート: GitHub でのリポジトリの使用 CodeCatalyst

以下の手順を実行して、GitHub リポジトリ拡張機能をインストールし、 GitHub アカウントに接続し、 GitHub CodeCatalyst リポジトリを既存のプロジェクトにリンクします。

GitHub リポジトリ拡張機能のインストール、 GitHub アカウントへの接続、 GitHub 新しいプロジェクトの作成時にリポジトリのリンクを行うこともできます。 CodeCatalyst 詳細については、「GitHubリポジトリをリンクしてプロジェクトを作成する」を参照してください。

目次

ステップ 1: GitHub カタログからエクステンションをインストールする CodeCatalyst

GitHub CodeCatalyst でリポジトリを使用する最初の手順は、GitHub カタログからリポジトリ拡張機能をインストールすることです。 CodeCatalyst エクステンションをインストールするには、GitHub リポジトリエクステンションを選択して以下の手順を実行します。

> **Important**
>
> GitHub リポジトリ拡張のインストールと設定の一環として、 GitHub アカウントに拡張機能をインストールする必要があります。これを行うには、 GitHub CodeCatalyst アカウント管理者とスペース管理者の両方である必要があります。

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/ 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。
3. CodeCatalyst CodeCatalyst

   トップメニューのカタログアイコンを選択してカタログに移動します。GitHub リポジトリを検索したり、カテゴリに基づいてエクステンションを絞り込んだりできます。
4. (オプション) エクステンションに付与される権限など、エクステンションに関する詳細情報を表示するには、GitHub リポジトリエクステンション名を選択します。
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。

リポジトリ拡張機能をインストールすると、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、GitHub GitHub 接続されているアカウントとリンクされたリポジトリを表示および管理できます。 GitHub

ステップ 2: GitHub CodeCatalyst アカウントをスペースConnect する

GitHub リポジトリ拡張機能をインストールしたら、 GitHub CodeCatalyst 次のステップはアカウントをスペースに接続することです。

> **Important**
>
> CodeCatalyst アカウントをスペースに接続するには、 GitHub GitHub CodeCatalyst アカウント管理者とスペース管理者の両方である必要があります。

GitHub アカウントを以下に接続するには CodeCatalyst

1. [Connect GitHub 済みアカウント] タブで、[GitHub アカウントを接続] を選択して外部サイトに移動します GitHub。
2. GitHub GitHub 認証情報を使用して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**
   >
   > GitHub 以前にアカウントを別のスペースに接続したことがある場合は、再認証を求めるメッセージは表示されません。代わりに、複数のアカウントのメンバーまたはコラボレーターである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、 GitHub CodeCatalyst 1つのアカウントにのみ所属している場合はAmazonアプリケーションの設定ページが表示されます。 GitHub 許可したいリポジトリへのアクセスをアプリケーションに設定し、[Save] を選択します。で GitHub [保存] ボタンがアクティブになっていない場合は、設定を変更してからやり直してください。

3. 現在およびfuture CodeCatalyst すべてのリポジトリへのアクセスを許可するか、 GitHub 使用したい特定のリポジトリを選択するかを選択します。 CodeCatalyst GitHub デフォルトオプションはアカウント内のすべてのリポジトリです。 GitHub
4. に与えられた権限を確認し CodeCatalyst、[インストール] を選択します。

GitHub アカウントに接続すると CodeCatalyst、GitHub リポジトリ拡張の詳細ページの [GitHub アカウント] タブに接続されたアカウントを確認できます。

ステップ 3: GitHub リポジトリをプロジェクトにリンクする CodeCatalyst

GitHub でリポジトリを使用する 3 CodeCatalyst つ目の最後のステップは、 CodeCatalyst 使用したいプロジェクトにリポジトリをリンクすることです。

> **Important**
>
> GitHub リポジトリはコントリビューターとしてリンクできますが、 GitHub 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だけです。詳細については、「 GitHub リポジトリをプロジェクトからリンク解除する CodeCatalyst 」を参照してください。

GitHub CodeCatalyst GitHub リポジトリ拡張の詳細ページからリポジトリをプロジェクトにリンクするには

1. 「リンクされたリポジトリ」タブで、「 GitHub リポジトリをリンク GitHub 」を選択します。
2. GitHub アカウントドロップダウンメニューから、 GitHub リンクするリポジトリを含むアカウントを選択します。
3. GitHub リポジトリのドロップダウンメニューから、 CodeCatalyst プロジェクトにリンクしたい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されている場合、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 プロジェクトドロップダウンメニューから、 CodeCatalyst GitHub リポジトリをリンクしたいプロジェクトを選択します。
5. [Link (リンク)] を選択します。

Code GitHub のソースリポジトリから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。詳細については、「 GitHub でのリポジトリの管理 CodeCatalyst」を参照してください。

次のステップ

GitHub リポジトリ拡張機能をインストールし、 GitHub アカウントを接続し、 GitHub CodeCatalyst リポジトリを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。詳細については、「 GitHub CodeCatalyst ワークフローでのリポジトリの使用」および「開発環境の作成」を参照してください。

GitHub でのアカウントの管理 CodeCatalyst

GitHub でリポジトリを使用するには CodeCatalyst、 GitHub アカウントをスペースに接続する必要があります。 CodeCatalyst でアカウントのリポジトリを使用する必要がなくなった場合は CodeCatalyst、そのアカウントの接続を解除できます。 GitHub アカウントが切断されると、そのアカウントのリポジトリ内のイベントはワークフローの実行を開始しなくなり、それらのリポジトリを Dev Environments で使用できなくなります。 CodeCatalyst

> **Important**
>
> GitHub アカウントとスペースを接続または接続解除するには、 CodeCatalyst GitHub アカウント管理者とスペース管理者の両方である必要があります。 CodeCatalyst

目次

GitHub CodeCatalyst アカウントをスペースConnect する

GitHub CodeCatalyst アカウントをスペースに接続するには、次の手順を実行してください。

GitHub アカウントをに接続するには 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。
3. 以下のいずれかを実行して、GitHub スペースにインストールされているリポジトリ拡張機能を表示します。

a. [設定] を選択し、次に [インストール済みの拡張機能] を選択します。

b. 

トップメニューのカタログアイコンを選択します。

4. GitHub リポジトリで [設定] を選択します。
5. [Connect GitHub 済みアカウント] タブで、[GitHub アカウントを接続] を選択して外部サイトに移動します GitHub。
6. GitHub GitHub 認証情報を使用して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**
>
> GitHub 以前にアカウントをスペースに接続したことがある場合は、再認証を求めるメッセージは表示されません。代わりに、複数のスペースのメンバーまたはコラボレーターである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、 GitHub CodeCatalyst 1つのスペースにのみ所属している場合はAmazonアプリケーションの設定ページが表示されます。 GitHub 許可したいリポジトリへのアクセスをアプリケーションに設定し、[Save] を選択します。[保存] ボタンがアクティブになっていない場合は、設定を変更してからやり直してください。

7. 現在およびfuture CodeCatalyst すべてのリポジトリへのアクセスを許可するか、 GitHub 使用したい特定のリポジトリを選択するかを選択します。 CodeCatalyst GitHub デフォルトオプションはスペース内のすべてのリポジトリです。 GitHub
8. に与えられた権限を確認し CodeCatalyst、[インストール] を選択します。

リポジトリ拡張機能をインストールすると、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、GitHub GitHub GitHub 接続されているアカウントとリンクされているリポジトリを表示および管理できます。

GitHub CodeCatalyst プロジェクト作成時にアカウントをスペースConnect する

CodeCatalyst 新しいプロジェクトを作成するときに、 GitHub CodeCatalyst アカウントをスペースに接続できます。詳細については、「GitHubリポジトリをリンクしてプロジェクトを作成する」を参照してください。

GitHub アカウントをスペースから切断します。 CodeCatalyst

GitHub アカウントをスペースから切り離すには、次の手順を実行します。 CodeCatalyst アカウントが切断されると、そのアカウントのリポジトリ内のイベントはワークフローの実行を開始しなくなり、それらのリポジトリを Dev Environments で使用できなくなります。 CodeCatalyst

> **Note**
>
> アカウントを切断するには、まず、 GitHub そのアカウントからリンクされているすべてのリポジトリのリンクを解除する必要があります。 GitHub 詳細については、「GitHub リポジトリをプロジェクトからリンク解除する CodeCatalyst 」を参照してください。

アカウントとの接続を解除するには GitHub

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。
3. 以下のいずれかを実行して、GitHub CodeCatalyst スペースにインストールされているリポジトリ拡張機能を表示します。

   a. [設定] を選択し、次に [インストール済みの拡張機能] を選択します。

   b. 

      プメニューのカタログアイコンを選択します。

4. GitHub リポジトリで [設定] を選択します。
5. GitHub アカウントタブで、 GitHub 接続を解除したいアカウントを選択します。
6. [アカウントの接続解除 GitHub] を選択します。
7. 「接続解除」ダイアログ・ボックスで、アカウントの接続解除による影響を確認します。
8. 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。

トッ

GitHub でのリポジトリの管理 CodeCatalyst

GitHub でリポジトリを使用するには CodeCatalyst、まずリポジトリをプロジェクトにリンクする必要があります。 CodeCatalyst GitHub リポジトリをリンクする前に、 GitHub リポジトリが属するアカウントをスペースに接続する必要があります。 CodeCatalyst 詳細については、「GitHub CodeCatalyst アカウントをスペースConnect する」を参照してください。

GitHub でリポジトリを使用する必要がなくなった場合は CodeCatalyst、 CodeCatalyst プロジェクトからそのリポジトリをリンク解除できます。リポジトリのリンクが解除されると、そのリポジトリ内のイベントはワークフローの実行を開始しなくなり、そのリポジトリを CodeCatalyst Dev Environments で使用できなくなります。

目次

GitHub リポジトリをプロジェクトにリンクします。 CodeCatalyst

GitHub リンクされたリポジトリをワークフローで使用できます。 GitHub リンクされたリポジトリ内のイベントによって、ワークフローの設定に応じてコードをビルド、テスト、またはデプロイするワークフローが開始されます。リンクされた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、 GitHub リンクされたリポジトリに保存されます。 GitHub GitHub リンクされたリポジトリを Dev Environments 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 GitHub CodeCatalyst リポジトリをプロジェクトにリンクするには、リポジトリ拡張機能の詳細ページ、またはプロジェクト自体のコードのソースリポジトリビューから行えます。GitHub

> **Note**
>
> GitHub リポジトリはスペース内の 1 CodeCatalyst つのプロジェクトにのみリンクできます。プロジェクト内の別のリポジトリと同じ名前のリポジトリをリンクすることはできません。

GitHub CodeCatalyst GitHub リポジトリ拡張の詳細ページからリポジトリをプロジェクトにリンクするには

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。
3. 以下のいずれかを実行して、GitHub スペースにインストールされているリポジトリ拡張機能を表示します。

   a. [設定] を選択し、[インストール済みの拡張機能] を選択します。

b. 

プメニューのカタログアイコンを選択します。

トッ

4. GitHub リポジトリで [設定] を選択します。
5. [リンクされたリポジトリ] タブで [GitHub リポジトリをリンク GitHub] を選択します。
6. GitHub アカウントドロップダウンから、 GitHub リンクしたいリポジトリを含むアカウントを選択します。
7. GitHub リポジトリードロップダウンから、プロジェクトにリンクしたいリポジトリを選択します。 CodeCatalyst

> **Tip**
>
> リポジトリの名前がグレー表示されている場合、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。

8. (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 アプリケーションのリポジトリアクセス用に設定されていない可能性があります。 GitHub接続したアカウントで CodeCatalyst 、 GitHub どのリポジトリを使用できるかを設定できます。

   a. GitHubアカウントに移動して [設定] を選択し、[アプリケーション] を選択します。
   b. 「 GitHub インストール済みアプリ」タブで、Amazon CodeCatalyst アプリケーションの「設定」を選択します。
   c. 以下のいずれかを実行して、 GitHub リンクしたいリポジトリへのアクセスを設定します。 CodeCatalyst

      - 現在およびfuture すべてのリポジトリへのアクセスを提供するには、[All repositories] 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、[リポジトリのみ選択] を選択し、[リポジトリを選択] ドロップダウンを選択してから、リンクを許可するリポジトリを選択します。 CodeCatalyst

> **Note**
>
> - GitHub 空のリポジトリやアーカイブされたリポジトリはプロジェクトでは使用できません。 CodeCatalyst

- GitHub プロジェクト内のリポジトリと同じ名前のリポジトリをリンクすることはできません。 CodeCatalyst
- GitHub リポジトリエクステンションは GitHub Enterprise Server のリポジトリと互換性がありません。

9. CodeCatalyst プロジェクトドロップダウンメニューから、 CodeCatalyst GitHub リポジトリをリンクするプロジェクトを選択します。
10. [Link (リンク)] を選択します。

GitHub CodeCatalyst プロジェクトのソースリポジトリページからリポジトリをプロジェクトにリンクするには

1. https://codecatalyst.aws/ CodeCatalyst で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. [リポジトリを追加] を選択し、[リポジトリをリンク] 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、を選択しますGitHub。
6. GitHub アカウントドロップダウンメニューから、 GitHub リンクするリポジトリを含むアカウントを選択します。
7. GitHub リポジトリのドロップダウンメニューから、 GitHub CodeCatalyst プロジェクトをリンクしたい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されている場合、そのリポジトリはすでに Amazon CodeCatalyst 内の別のプロジェクトにリンクされているため、リンクできません。

8. (オプション) GitHub リポジトリのドロップダウンにリポジトリが表示されない場合は、GitHub そのリポジトリがスペースアプリケーションのリポジトリアクセス用に設定されていない可能性があります。 GitHub GitHub CodeCatalyst 接続したアカウントでどのリポジトリを使用できるかを設定できます。

   a. GitHubアカウントに移動して [設定] を選択し、[アプリケーション] を選択します。
   b. 「 GitHub インストール済みアプリ」タブで、Amazon CodeCatalyst アプリケーションの「設定」を選択します。

c. 以下のいずれかを実行して、 GitHub リンクしたいリポジトリへのアクセスを設定します。 CodeCatalyst

- 現在およびfuture すべてのリポジトリへのアクセスを提供するには、[All repositories] 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、[リポジトリのみ選択] を選択し、[リポジトリを選択] ドロップダウンを選択してから、リンクを許可するリポジトリを選択します。 CodeCatalyst

> **Note**
>
> - GitHub 空のリポジトリやアーカイブされたリポジトリはプロジェクトでは使用できません。 CodeCatalyst
> - GitHub プロジェクト内のリポジトリと同じ名前のリポジトリをリンクすることはできません。 CodeCatalyst
> - GitHub リポジトリエクステンションは GitHub Enterprise Server のリポジトリと互換性がありません。

9. [Link (リンク)] を選択します。

GitHub CodeCatalyst プロジェクト作成中にリポジトリをプロジェクトにリンクする

新しいプロジェクトを作成するときに、 GitHub CodeCatalyst リポジトリを新しいプロジェクトにリンクできます。 CodeCatalyst 詳細については、「GitHubリポジトリをリンクしてプロジェクトを作成する」を参照してください。

GitHub リポジトリをプロジェクトからリンク解除する CodeCatalyst

リポジトリのリンクを解除しても、 GitHub 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、リポジトリのリンクを解除すると、 GitHub そのリポジトリ内のイベントはワークフローの実行を開始しなくなり、そのリポジトリを Dev Environments で使用できなくなります。

> **Important**
>
> GitHub リポジトリをプロジェクトからリンク解除するには、 CodeCatalyst 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。

リポジトリのリンクを解除するには GitHub

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。
3. 以下のいずれかを実行して、GitHub スペースにインストールされているリポジトリ拡張機能を表示します。

   a. [設定] を選択し、[インストール済みの拡張機能] を選択します。

   b. 

   プメニューのカタログアイコンを選択します。

トッ
4. GitHub リポジトリで [設定] を選択します。
5. 「 GitHub リンクされたリポジトリ」タブで、 GitHub リンクを解除するリポジトリを選択します。
6. [リポジトリをリンク解除 GitHub] を選択します。
7. 「リンク解除」ダイアログで、リポジトリのリンク解除による影響を確認します。
8. テキスト入力フィールドに unlink と入力し、[リンク解除] を選択します。

GitHub でリンクされたリポジトリを表示する CodeCatalyst

GitHub リンクされたリポジトリは、GitHub 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拡張機能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選択しても、そのリポジトリは開かれません。 CodeCatalyst代わりにで開き GitHub、リンク先のリポジトリ内のコードを表示して作業できます。

GitHub リンクされたリポジトリをで表示するには CodeCatalyst

1. https://codecatalyst.aws/ CodeCatalyst で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。

GitHub リンクされたリポジトリを自分から表示するには、拡張機能の詳細ページにアクセスしてください。

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. CodeCatalyst スペースに移動し、「インストール済みの拡張機能」タブを選択します。
3. GitHub リポジトリで [設定] を選択します。
4. 「 GitHub リンクされたリポジトリ」タブを選択すると、 GitHub CodeCatalyst スペース内のプロジェクトに接続されているすべてのリポジトリが表示されます。 CodeCatalyst

GitHub プロジェクトにリンクされているリポジトリがリストに表示されます。ファイルを表示、GitHub 編集するリポジトリを選択します。 GitHub

> **Note**
>
> GitHub ワークフローがソースアクションでリポジトリを使用する場合、ビジュアルエディターまたはの YAML エディターでワークフロー YAML に加えた変更は自動的にコミットされ、リポジトリにプッシュされます。 CodeCatalyst GitHub

GitHub CodeCatalyst ワークフローでのリポジトリの使用

GitHub リンクされた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。 GitHub リンクされたリポジトリ内の特定のブランチに変更を加えると、自動的にワークフローの実行が開始されます。

ワークフローは、継続的インテグレーションと継続的デリバリー (CI/CD) システムの一部としてコードをビルド、テスト、デプロイする方法を記述した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、つまりアクションを定義します。ワークフローでは、ワークフローを開始させるイベント、つまりトリガーも定義されます。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エディターまたは YAML エディター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、ブループリントを使用してプロジェクトを作成してください。各ブループリントには、レビュー、実行、実験が可能な、機能するワークフローが導入されています。

GitHub リンクされた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、 GitHub ワークフロー設定ファイルはその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。ワークフロー定義リファレンス

ワークフロー設定ファイルは、`./codecatalyst/workflows/` GitHub リポジトリ内の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ー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、「でのワークフロー入門 CodeCatalyst」および「ソースの操作」を参照してください。

GitHub リポジトリイベントの後にワークフローを自動的に開始します。

GitHub リポジトリの指定されたブランチにコードがプッシュされたときに、 CodeCatalyst 自動的に実行を開始するように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、`Triggers`ワークフロー設定ファイルの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任意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の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任意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の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、「トリガーの使用」を参照してください。

GitHub エンタープライズクラウドでの IP アドレスアクセス制限の使用

Amazon CodeCatalyst GitHub リポジトリエクステンションは、GitHub エンタープライズクラウド IP アクセス制限に対応しています。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 GitHubアプリケーションによる許可リストの設定を有効にして、その IP CodeCatalyst アドレスをに自動的に登録できるようにすることもできます。または、 CodeCatalyst IP アドレスを手動で追加することもできます。詳細については、「 GitHub リポジトリエクステンションが使用する IP アドレス」を参照してください。

CodeCatalyst GitHub IPアドレスが組織の許可リストに含まれていない場合、 CodeCatalyst GitHub GitHub Amazonアプリはリポジトリにアクセスできません。

GitHub リポジトリエクステンションが使用する IP アドレス

GitHub リポジトリエクステンションは次の IP GitHub アドレスを使用して組織のリソースにアクセスします。

```
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
```

GitHub ワークフローが失敗したときにプルリクエストのマージをブロックする

GitHub リポジトリをにリンクすると CodeCatalyst、 CodeCatalyst プルリクエスト用のワークフローを追加できます。特定のコミットで 1 つ以上のワークフローが実行される可能性があり、CodeCatalyst 内の各ワークフローの実行ステータスもコミットステータスの一部として反映されます GitHub。新しいコミットがプッシュされると、 GitHub その新しいコミットの新しいワークフロー実行ステータスが反映されます。あるコミットのワークフローを再実行すると、新しいワークフローの実行ステータスが、そのコミットとワークフローの以前のステータスよりも優先されます。

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、 GitHub プルリクエストのマージをブロックするようにブランチ保護ルールを設定できます。ブランチ保護ルールでは、 GitHub最新のコミットのステータスがプルリクエストをマージできるかどうかに影響します。詳細については、「ステータスチェックについて」と「保護されたブランチについて」 GitHub のドキュメントを参照してください。ワークフローの詳細については、「」ランの処理 と「」を参照してくださいトリガーの使用。

での Jira 課題の使用 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、管理するのに役立つソフトウェアアプリケーションです。Jira ソフトウェアエクステンションを使用すると、Amazon CodeCatalyst プロジェクトで Jira プロジェクトを使用できます。

> **Note**
>
> CodeCatalyst Jira Software クラウドとのみ互換性があります。

Amazon CodeCatalyst プロジェクト用に Jira Software エクステンションをインストールして設定すると、次のことができるようになります。

1. から CodeCatalyst Jira プロジェクトにアクセスするには、Jira プロジェクトをプロジェクトにリンクします。 CodeCatalyst
2. Jira CodeCatalyst の課題をプルリクエストで更新
3. Jira CodeCatalyst 課題内のリンクされた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

以下のトピックでは、Jira Software エクステンションをインストールし、で Jira を使用するように設定する方法について説明します。 CodeCatalyst

トピック

- クイックスタート:での Jira 課題の使用 CodeCatalyst
- での Jira サイトの管理 CodeCatalyst
- での Jira プロジェクトの管理 CodeCatalyst
- Jira CodeCatalyst 課題をプルリクエストにリンクする
- CodeCatalyst Jira でのイベントの表示
- で Jira 課題を検索する CodeCatalyst

クイックスタート:での Jira 課題の使用 CodeCatalyst

次の手順を実行して Jira Software エクステンションをインストールし、Jira CodeCatalyst サイトをスペースに接続し、Jira プロジェクトをプロジェクトにリンクします。 CodeCatalyst

目次

ステップ 1: カタログから Jira Software エクステンションをインストールする CodeCatalyst

で Jira を管理するための最初のステップは、カタログから Jira Software CodeCatalyst エクステンションをインストールすることです。 CodeCatalyst 拡張機能をインストールするには、Jira Software エクステンションを選択して次の手順を実行します。

> **Important**
>
> Jira Software エクステンションのインストールと設定の一環として、アトラシアンアカウントと既存の Jira サイトが必要ですが、Jira プロジェクトは後で作成できます。Jira サイト管理者およびスペース管理者である必要があります。 CodeCatalyst

カタログから拡張機能をインストールするには 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。
3. CodeCatalyst CodeCatalyst

   トップメニューのカタログアイコンを選択してカタログに移動します。Jira Software を検索したり、カテゴリに基づいてエクステンションをフィルタリングしたりできます。
4. (オプション) エクステンションに付与される権限など、エクステンションに関する詳細情報を表示するには、Jira Software エクステンション名を選択します。
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。

Jira Software エクステンションをインストールすると、Jira Software エクステンションの詳細ページに移動します。このページでは、接続されている Jira サイトやリンクされている Jira プロジェクトを表示および管理できます。

ステップ 2: Jira サイトをスペースConnect する CodeCatalyst

Jira Software エクステンションをインストールしたら、次のステップは Jira サイトをスペースに接続することです。 CodeCatalyst

> **Important**
>
> Jira CodeCatalyst サイトをスペースに接続するには、Jira サイト管理者とスペース管理者の両方である必要があります。 CodeCatalyst

Jira サイトを以下に接続するには CodeCatalyst

1. [Connect されている Jira サイト] タブで、[Jira サイトを接続] を選択して Atlassian Marketplace の外部サイトに移動します。
2. Jira CodeCatalyst サイトへのインストールを開始するには、[今すぐ入手] を選択します。
3. 役割に応じて、次のいずれかを実行してください。

    1. Jira サイト管理者の場合は、 CodeCatalyst サイトドロップダウンメニューからアプリケーションをインストールする Jira サイトを選択し、次に [アプリのインストール] を選択します。

        > **Note**
        >
        > Jira サイトが 1 つしかない場合、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、 CodeCatalyst サイトドロップダウンメニューからアプリケーションをインストールする Jira サイトを選択し、[アプリをリクエスト] を選択します。Jira アプリのインストールに関する詳細は、「アプリをインストールできるのは誰ですか?」を参照してください。 。

        b. インストールが必要な理由を入力テキストフィールドに入力するか、 CodeCatalyst デフォルトのテキストをそのまま使用して、[Submit request] を選択します。

4. CodeCatalyst アプリケーションのインストール時に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. アプリケーションがインストールされたら、[Return CodeCatalyst to] CodeCatalyst を選択して戻ります。

Jira サイトを接続すると CodeCatalyst、Jira Software 拡張機能の詳細ページの [接続されている Jira サイト] タブに接続サイトを表示できます。

ステップ 3: Jira プロジェクトをプロジェクトにリンクする CodeCatalyst

で Jira プロジェクトを管理する 3 CodeCatalyst 番目のステップは、Jira プロジェクトを、CodeCatalyst 使用したいプロジェクトにリンクすることです。

> **Note**
>
> 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 CodeCatalyst プロジェクトは複数のプロジェクトにリンクできます。

> **Important**
>
> Jira プロジェクトをプロジェクトにリンクするには、 CodeCatalyst CodeCatalyst CodeCatalyst 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。

Jira Software 拡張機能の詳細ページから Jira CodeCatalyst プロジェクトをプロジェクトにリンクするには

1. [リンクされた Jira プロジェクト] タブで [Jira プロジェクトをリンク] を選択します。
2. Jira サイト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 プロジェクトドロップダウンメニューから、プロジェクトにリンクしたいプロジェクトを選択します。 CodeCatalyst
4. CodeCatalyst プロジェクト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 プロジェクトがプロジェクトにリンクされると、 CodeCatalyst 課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課題は Jira CodeCatalyst プロジェクトにリンクする Jira 課題アイテムに置き換えられます。

ステップ 4: Jira 課題をプルリクエストにリンクする CodeCatalyst

Jira プロジェクトをプロジェクトにリンクしたら、 CodeCatalyst CodeCatalyst プルリクエストを作成して課題をリンクし、プルリクエストのプロパティとして表示させることができます。

> **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することはできません。プルリクエストの詳細については、の「プルリクエストの処理」を参照してください CodeCatalyst。

Jira CodeCatalyst の課題をプルリクエストにリンクするには

1. 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。
2. [プルリクエストを作成] を選択し、プルリクエストの詳細を入力します。
3. 「ソースリポジトリ」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。
4. 「ソースブランチ」ドロップダウンメニューから、レビューしたい変更を含むブランチを選択します。
5. 宛先ブランチのドロップダウンメニューから、レビューした変更をマージするブランチを選択します。
6. プルリクエストのタイトル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。
7. Jira 課題のリンク-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、リンクされている Jira プロジェクトから追加したい Jira 課題を検索します。
8. プルリクエストに追加する Jira 課題を選択します。
9. [Create] を選択してプルリクエストを作成します。

CodeCatalyst プルリクエストの概要ステータスと関連するワークフローイベントのステータスが Jira 課題に反映されます。

次のステップ

Jira Software エクステンションをインストールし、Jira サイトに接続し、Jira プロジェクトを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、からの更新が Jira CodeCatalyst プロジェクトに反映されます。 CodeCatalyst Jira CodeCatalyst でのイベントの表示に関する詳細は、「」を参照してください。 CodeCatalyst Jira でのイベントの表示

での Jira サイトの管理 CodeCatalyst

で Jira プロジェクトを管理するには CodeCatalyst、Jira サイトをスペースに接続する必要があります。 CodeCatalyst で Jira サイトのプロジェクトを使用したくない場合は CodeCatalyst、その Jira サイトの接続を切断できます。Jira サイトの接続が切断されると、サイトのプロジェクトの Jira 課題はプロジェクトで利用できなくなり、 CodeCatalyst CodeCatalyst 課題は再び課題プロバイダーになります。

> **Important**
>
> Jira サイトとスペースを接続または接続解除するには、Jira CodeCatalyst サイト管理者とスペース管理者の両方である必要があります。 CodeCatalyst

目次

Jira サイトをスペースに接続する CodeCatalyst

Jira サイトをスペースに接続するには、次の手順を実行します。 CodeCatalyst

Jira サイトを接続するには CodeCatalyst

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. CodeCatalyst スペースに移動します。
3. 以下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 Jira Software 拡張機能を表示します。

   a. [設定] を選択し、次に [インストール済みの拡張機能] を選択します。

b. 

トップメニューのカタログアイコンを選択します。

4. Jira Software で [設定] を選択します。
5. [Connect されている Jira サイト] タブで、[Jira サイトを接続] を選択して Atlassian Marketplace の外部サイトに移動します。
6. Jira CodeCatalyst サイトへのインストールを開始するには、[今すぐ入手] を選択します。

> **Note**
>
> 以前に Jira CodeCatalyst サイトにインストールしていた場合は、通知が届きます。[始める] を選択して最後のステップに進みます。

7. 役割に応じて、次のいずれかを実行します。

   1. Jira サイト管理者の場合は、 CodeCatalyst サイトドロップダウンメニューからアプリケーションをインストールする Jira サイトを選択し、次に [アプリのインストール] を選択します。

      > **Note**
      >
      > Jira サイトが 1 つしかない場合、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、 CodeCatalyst サイトドロップダウンメニューからアプリケーションをインストールする Jira サイトを選択し、[アプリをリクエスト] を選択します。Jira アプリのインストールに関する詳細は、「アプリをインストールできるのは誰ですか?」を参照してください。 。

      b. インストールが必要な理由を入力テキストフィールドに入力するか、 CodeCatalyst デフォルトのテキストをそのまま使用して、[Submit request] を選択します。

8. CodeCatalyst アプリケーションのインストール時に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。
9. アプリケーションがインストールされたら、[Return CodeCatalyst to] CodeCatalyst を選択して戻ります。

Jira サイトを接続すると CodeCatalyst、Jira Software 拡張機能の詳細ページの [接続されている Jira サイト] タブに接続サイトを表示できます。

Jira サイトをスペースから切断する CodeCatalyst

Jira サイトをスペースから切断するには、次の手順を実行します。 CodeCatalyst

> **Note**
>
> Jira サイトの接続を解除するには、まず、リンクされているすべての Jira プロジェクトをそのアカウントからリンク解除する必要があります。詳細については、「Jira プロジェクトをプロジェクトからリンク解除する CodeCatalyst」を参照してください。

Jira サイトの接続を解除するには

1. https://codecatalyst.aws/ CodeCatalyst でコンソールを開きます。
2. CodeCatalyst スペースに移動します。
3. 以下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 Jira Software 拡張機能を表示します。

   a. [設定] を選択し、次に [インストール済みの拡張機能] を選択します。

   b. 

   プメニューのカタログアイコンを選択します。

トッ

4. Jira Software で [設定] を選択します。
5. 接続されている Jira サイトタブで、接続を解除したい Jira サイトを選択します。
6. [Jira サイトの接続解除] を選択します。
7. 接続解除ダイアログボックスで、サイトを切断した場合の影響を確認します。
8. 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 Jira プロジェクトの管理 CodeCatalyst

で Jira プロジェクトを管理するには CodeCatalyst、まず Jira プロジェクトをプロジェクトにリンクする必要があります。 CodeCatalystその後、Jira CodeCatalyst 課題をプロジェクト内のプルリクエストに追加して使用することができます。 CodeCatalyst Jira プロジェクトをリンクする前に、その

プロジェクトが属する Jira サイトをスペースに接続する必要があります。 CodeCatalyst 詳細については、「Jira サイトをスペースに接続する CodeCatalyst」を参照してください。

Jira プロジェクトを使用する必要がなくなった場合は CodeCatalyst、そのプロジェクトをプロジェクトからリンク解除できます。 CodeCatalystJira プロジェクトのリンクが解除されると、Jira CodeCatalyst 課題はプロジェクトで利用できなくなり、 CodeCatalyst 課題は再び課題プロバイダーになります。

> **Important**
>
> Jira プロジェクトをプロジェクトにリンクするには、 CodeCatalyst CodeCatalyst 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。 CodeCatalyst

目次

Jira プロジェクトをプロジェクトにリンクする CodeCatalyst

リンクされた Jira プロジェクトを使用して課題を管理したり、 CodeCatalyst プルリクエストを Jira 課題にリンクしたりできます。 CodeCatalyst プルリクエストの概要ステータスと関連するワークフローイベントのステータスは Jira 課題に反映されます。

> **Note**
>
> 1 CodeCatalyst つのプロジェクトは 1 つの Jira プロジェクトにのみリンクできます。Jira CodeCatalyst プロジェクトは複数のプロジェクトにリンクできます。

Jira Software 拡張機能の詳細ページから Jira CodeCatalyst プロジェクトをプロジェクトにリンクするには

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。
3. 以下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 Jira Software 拡張機能を表示します。

a. [設定] を選択し、次に [インストール済みの拡張機能] を選択します。

b. 

トップメニューのカタログアイコンを選択します。

4. Jira Software で [設定] を選択します。
5. [リンクされた Jira プロジェクト] タブで [Jira プロジェクトをリンク] を選択します。
6. Jira サイト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。
7. Jira プロジェクト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プロジェクトを選択します。 CodeCatalyst
8. CodeCatalyst プロジェクト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。
9. [Link (リンク)] を選択します。

Jira プロジェクトがプロジェクトにリンクされると、 CodeCatalyst 課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課題は Jira CodeCatalyst プロジェクトにリンクする Jira 課題アイテムに置き換えられます。

Jira プロジェクトをプロジェクトからリンク解除する CodeCatalyst

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を含む Jira プロジェクトは削除されず、変更も加えられません。ただし、Jira プロジェクトのリンクを解除すると、そのプロジェクトの Jira 課題はプロジェクトにリンクできなくなります。 CodeCatalyst

Jira プロジェクトのリンクを解除するには

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。
3. 以下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 Jira Software 拡張機能を表示します。

a. [設定] を選択し、次に [インストール済みの拡張機能] を選択します。

b. 

トップメニューのカタログアイコンを選択します。

4. リンクされた 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ます。

5. [Jira プロジェクトのリンク解除] を選択します。
6. リンク解除ダイアログボックスで、リポジトリのリンク解除による影響を確認します。
7. テキスト入力フィールドに unlink と入力し、[リンク解除] を選択します。

Jira CodeCatalyst 課題をプルリクエストにリンクする

CodeCatalyst ソースリポジトリで作成されたプルリクエストを Jira の課題にリンクできます。Jira 課題をリンクすると、課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 課題に追加されます。プルリクエストは 1 つ以上の Jira 課題にリンクできます。 CodeCatalyst リンクできるのはソースリポジトリにあるプルリクエストのみで、 GitHubのようなサードパーティのリポジトリにあるプルリクエストはリンクできません。Jira の課題をプルリクエストにリンクする前に、Jira プロジェクトをプロジェクトにリンクする必要があります。 CodeCatalyst Jira プロジェクトを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。 CodeCatalyst での Jira サイトの管理 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することはできません。プルリクエストの詳細については、の「プルリクエストの処理」を参照してください CodeCatalyst。

Jira CodeCatalyst の課題をプルリクエストにリンクするには

1. https://codecatalyst.aws/ CodeCatalyst で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. [プルリクエストを作成] を選択し、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」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、レビューしたい変更を含むブランチを選択します。
7. 宛先ブランチのドロップダウンメニューから、レビューした変更をマージするブランチを選択します。

8. プルリクエストのタイトル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 課題のリンク-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、リンクされている Jira プロジェクトから追加したい Jira 課題を検索します。
10. プルリクエストに追加する Jira 課題を選択します。
11. [Create] を選択してプルリクエストを作成します。

Jira CodeCatalyst の課題を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、リンクされた課題、必須のレビュー担当者、任意のレビュー担当者、および作成者が含まれます。

> **Note**
>
> Jira 課題に関連する担当者と作成者の情報はでは利用できません。 CodeCatalyst

プルリクエストをリンクすると、 CodeCatalyst 同期されたプロジェクトと Jira CodeCatalyst プロジェクトで更新内容を Jira プロジェクトに反映できるようになります。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 課題を Jira で表示すると表示されます。Jira CodeCatalyst でのイベントの表示に関する詳細は、「」を参照してください。 CodeCatalyst Jira でのイベントの表示

CodeCatalyst Jira でのイベントの表示

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、 CodeCatalyst プルリクエストの概要ステータスと関連するワークフローイベントのステータスが Jira 課題に反映されます。たとえば、プルリクエストをクローズまたはマージすると CodeCatalyst、ステータスの更新は Jira 課題に反映されます。 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、ワークフローが正常に実行されると Jira 課題にも送信されます。

Jira CodeCatalyst 課題内のイベ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 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] を選択してから、Jira プロジェクトで表示したい Jira 課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したい Jira 課題を選択します。

5. Jira プロジェクトの詳細ペインから、開発用にリストされているプルリクエストを選択すると、プルリクエストの詳細が表示されます。
6. (オプション) 最新のビルドを確認するには、[ビルド] タブを選択します。
7. (オプション) 開発状況を確認するには、「Deployments」タブを選択します。

で Jira 課題を検索する CodeCatalyst

Jira プロジェクトをリンクしたら、 CodeCatalyst グローバル検索バーを使用して、リンクされた Jira プロジェクトの課題を検索できます。 CodeCatalyst プルリクエストの課題にリンクしている間に Jira の課題を検索することもできます。Jira CodeCatalyst 課題をプルリクエス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。Jira CodeCatalyst 課題をプルリクエストにリンクする

リンクされている Jira プロジェクト内の Jira 課題を検索するには

1. https://codecatalyst.aws/ CodeCatalyst で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、リンクされている Jira プロジェクトから、プルリクエストにリンクしたい課題または Jira 課題を検索します。

で検索 CodeCatalyst

検索バーまたは専用の検索結果ウィンドウを使用して、コード、課題、プロジェクト、CodeCatalyst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と、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) CodeCatalyst の検索クエリ言語を使用して検索クエリを絞り込みます。詳細については、「検索クエリを絞り込む」を参照してください。
3. 次のいずれかを実行します。
   - 現在参加しているプロジェクト内のリソースを検索するには、「This project」を選択します。
   - 現在参加してい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. 以下のいずれかを実行して、検索結果を専用の検索結果ウィンドウに表示します。
   - クイック検索結果ウィンドウの下部で、「すべての結果をプロジェクト名 | space-name で表示」を選択すると、すべての検索結果が表示されます。
   - Enter キーを押すと、すべての検索結果が表示されます。

> **Tip**
>
> プルリクエストのコメントや説明、または課題のコメントや説明の中で、@ 記号の後に表示名やユーザー名を付けて、他のプロジェクトユーザーをメンションします。@ 記号の後に

Issue またはコードファイルの名前を付けることで、Issue 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。

検索クエリを絞り込む

検索しても探しているものが見つからない場合は、 CodeCatalystの専用クエリ言語で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、クエリ全体では 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

検索範囲を特定の種類の情報に絞り込むには、*type:result-type*検索に含めます。*result-type* は、、、またはですcode。issue project user

例:

- `type:code AND java`—「java」を含むコード関連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`—「Bug」を含む課題関連フィールドに課題結果を表示します。

  詳細については、「課題フィールド」を参照してください。
- `type:user AND 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。MaryMajor

  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

検索範囲を特定のフィールドに絞り込むには、検索に含めます*field-name:query*。ここで *field-name* はtitle、username、projectdescription、などで、*query* は検索対象のテキストです。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「bug」が含まれている結果を表示します。
- `username:John`— ユーザー名に「John」が含まれる結果を表示します。
- `project:DataFeeder`— プロジェクト「DataFeeder」の結果を表示します。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」が含まれる結果を表示します。

ブーリアン演算子による調整

検索フレーズの制約を指定するには、ブール演算子、、、を使用できます。AND OR NOT複数のフレーズをリストした場合、 CodeCatalystORデフォルトではでそれらを結合します。検索語句は括弧を使っ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、「README.md」のパスの結果を表示します。
- `buildspec.yml AND (repo:ServerlessAPI OR ServerlessWebApp)`— リポジトリが「サーバーレス API」または「」である「buildspec.yml」の結果を表示します。ServerlessWebApp
- `path:java NOT (path:py OR path:ts)`— パスに「java」が含まれるが「py」や「ts」は含まれていない場合の結果を表示します。

プロジェクトごとに絞り込む

検索範囲を特定のプロジェクトに絞り込むには、*project:name AND query*検索に含めます。*name* は検索対象のプロジェクト、*query* は検索するコンテンツです。

- `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

コンテンツ更新の遅延 — 名前の変更や課題の再割り当てなどのコンテンツ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 — 以下の特殊文字は、検索クエリで特に考慮する必要があります:. + - & & || ! ( ) { } [ ] ^ " ~ * ? : \ 特殊文字はクエリに影響しないため、削除するかエスケープする必要があります。文字をエスケープするには、その文字の前にバックスラッシュ (\) を追加します。たとえば、検索クエリ [Feature] は Feature か\ [Feature\] のどちらかでなければなりません。

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。すべて小文字で検索すると、大文字と小文字が変わってもクエリで単語が分割されるのを防ぐことができます。たとえば、MyService and のみを検索する場合はMyService、myservice結果にまたはのみが含まれないようにすることを検討してください。my service

デフォルトでは、検索では単語と単語の一部を OR 単位の接続詞で結合します。new functionたとえば、newfunctionとの両方を含む結果と、またはのみの結果を返す場合があります。new function後者を避けるには、複数の単語をと組み合わせてくださいAND。たとえば、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での最新のコミット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のコードを見つけるには、リポジトリをローカルに複製するか、Dev Environment でブランチを開くか、UI でブランチと詳細を確認することを検討してください。 CodeCatalyst デフォルトブランチを変更すると、検索で検出可能なファイルが更新されます。詳細については、「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」を参照してください。

検索可能なフィールドリファレンス

CodeCatalyst 検索クエリを入力すると、以下のフィールドが検索されます。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
ブランチ名	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。

フィールド	エイリアス	説明
コード	該当なし	ソースコードの中で検索に一致した部分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。で指定したブランチ名の先頭にあるコミット ID でも、そうでない場合もあります。branchName
[コミット/メッセージ]	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。で指定したブランチ名の先頭に表示されるコミットメッセージでも、そうでない場合もあります。branchName コミットメッセージが提供されなかった場合、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。ユーザー名が利用できない場合、この値は Git 設定ファイルで設定されているユーザーのメールアドレスになります。

フィールド	エイリアス	説明
lastUpdatedByID	該当なし	システムによって生成されたコードファイルを最後に更新したユーザーの固有 ID。ユーザー ID がない場合、この値はユーザーの電子メールアドレスである可能性があります。
lastUpdatedTime	該当なし	コードファイルを含むコミットで検索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)。
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固有 ID。
projectName	プロジェクト名、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。
リポジトリ ID	レポイド	システムによって生成されたソースリポジトリの固有 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。

課題フィールド

フィールド	エイリアス	説明
担当者 ID	譲受人 ID	課題に割り当てられたユーザーの、システム生成の固有 ID。

フィールド	エイリアス	説明
譲受人	担当者	課題に割り当てられたユーザーのユーザー名。
作成者	該当なし	課題を作成したユーザーの名前を表示します。
createdById	該当なし	システムによって生成された課題を作成したユーザーの固有 ID。
createdTime	該当なし	課題が作成された時刻 (協定世界時 (UTC) タイムスタンプ)。
description	該当なし	問題の説明。
はアーカイブされました	archived	課題をアーカイブ状態で作成するかどうかを示すブール値。
IsBlocked	ブロック済み	課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。
LabelID	ラベル ID	システムによって生成された課題のラベルの固有 ID。
lastUpdatedBy	該当なし	課題を最後に更新したユーザーの名前を表示します。
lastUpdatedByID	該当なし	問題を最後に更新したユーザーの、システム生成の固有 ID。
lastUpdatedTime	該当なし	課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)。

フィールド	エイリアス	説明
優先度	該当なし	課題の優先度 (割り当てられている場合)。
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固有 ID。
projectName	プロジェクト名、プロジェクト	この問題が確認されているプロジェクト。
ショート ID	該当なし	問題の短縮された自動インクリメント識別子。
ステータス	該当なし	課題のステータス。課題がバックログにあるのか、ボード上の列にあるのかを示します。
ステータス ID	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
description	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)。
projectName	プロジェクト	スペース内のプロジェクトの名前。

フィールド	エイリアス	説明
プロジェクトパス	該当なし	プロジェクト作成時に定義される、URL でルーティング可能なプロジェクト名。プロジェクト名を必要とする URL 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	ユーザーがサインアップしたときに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用の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、「ブランチを削除するには (コンソール)」および「コントリビューターロール」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、「ソースリポジトリを削除する」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、「Amazon での個人アクセストークンの管理 CodeCatalyst」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、「AWS招待の承諾とビルダー ID の作成」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、「Amazon での個人アクセストークンの管理 CodeCatalyst」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、「プロジェクト内のメンバーを表示する」および「Amazon のロールを使った作業 CodeCatalyst」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、「ブランチの許可されたアクションをブランチルールで管理」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の処理 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、「Amazon での個人アクセストークンの管理 CodeCatalyst」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「ファイルを表示する」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、「プルリクエストをマージする」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。

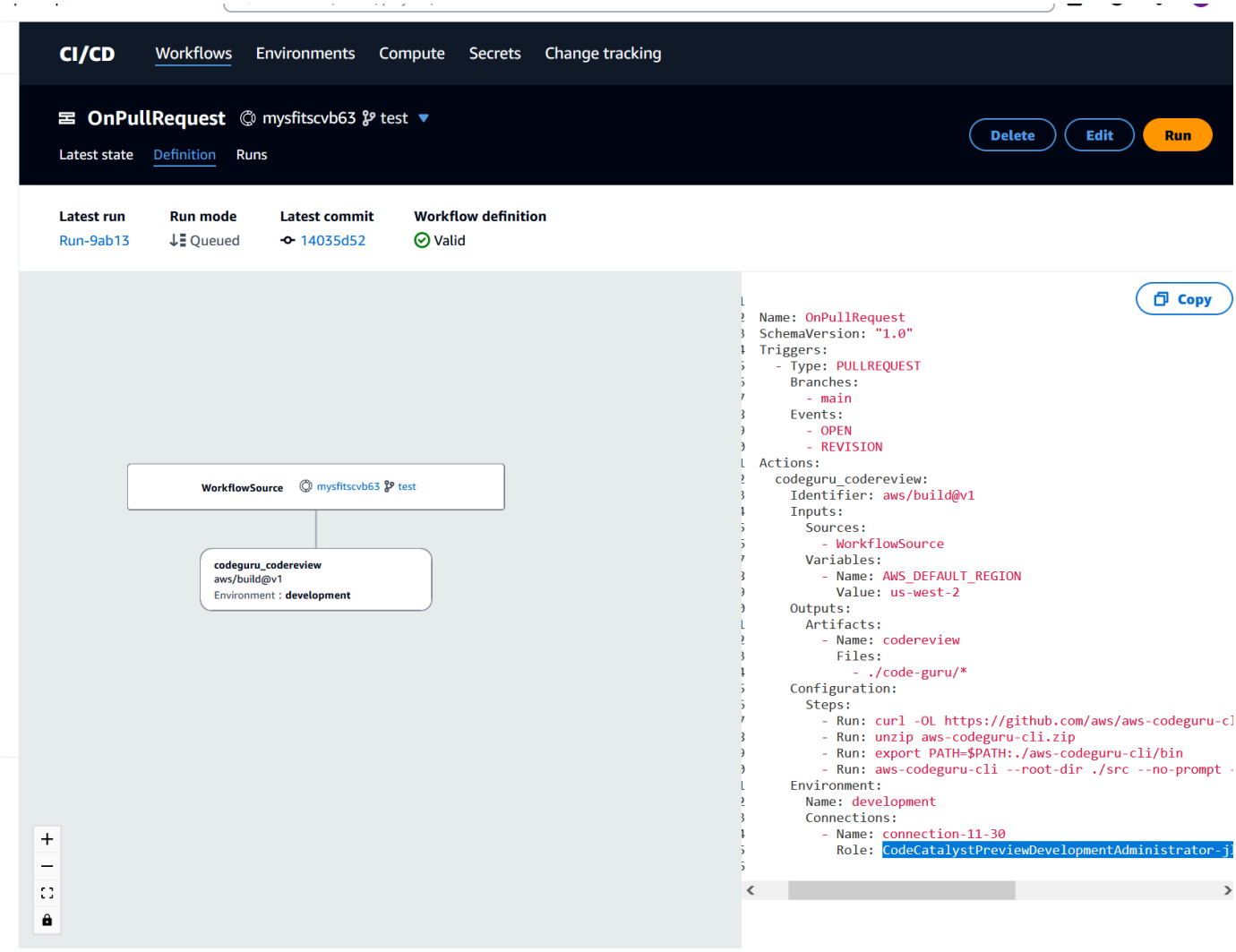
各プロジェクトには、AWS アカウントでプロジェクトに接続されているから提供されるロールと環境を使用するアクションを含むワークフローが含まれています。 CodeCatalystアクションと指定されたポリシーを含むワークフローは、ソースリポジトリの /.codecatalyst/workflows ディレクトリに保存されます。既存のワークフローに新しいロール ID を追加しない限り、ワークフロー YAML を変更する必要はありません。YAML テンプレート要素とフォーマットについては、を参照してください。ワークフロー定義リファレンス

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 CodeCatalyst のワークフロー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。ワークフローの詳細については、「???」を参照してください。

トピック

- 「ワークフローは無効です」というメッセージを修正する方法を教えてください。
- 「ワークフロー定義にエラーはありません」というエラーを修正する方法を教えてください。

「ワークフローは無効です」というメッセージを修正する方法を教えてください。

問題: CodeCatalyst コンソールの CI/CD の [ワークフロー] に、ワークフローが次のメッセージとともに表示されます。

```
Workflow is inactive.
```

このメッセージは、ワークフロー定義ファイルに、現在作業しているブランチには適用されないトリガーが含まれていることを示しています。たとえば、PUSHワークフロー定義ファイルに自分のブランチを参照するトリガーが含まれていても、mainそのブランチはフィーチャーブランチを使っているとします。フィーチャーブランチで行っている変更は適用されず、ワークフローの実行も開始されないためmain、 CodeCatalyst ブランチ上のワークフローをデコミッションしmain、Inactiveとしてマークします。

解決方法:

フィーチャーブランチでワークフローを開始したい場合は、次のことを行ってください。

- フィーチャーブランチのワークフロー定義ファイルで、BranchesTriggers以下のようになるようにセクションからプロパティを削除します。

  ```
  Triggers:
    - Type: PUSH
  ```

  この設定により、フィーチャー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。 CodeCatalyst トリガーがアクティブ化されると、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の実行が開始されます。
- フィーチャー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。
- フィーチャーブランチのワークフロー定義ファイルで、別のブランチ (例えば) PUSH ではなく自分のフィーチャーブランチを参照するようにセクションを変更します。main

> **Important**
>
> mainこれらの変更をマージして自分のブランチに戻すつもりがなければ、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に関する詳細は、を参照してくださいワークフローを編集するには。

トリガーについての詳細は、「トリガーの使用」を参照してください。

##

問題:以下のいずれかのエラーメッセージが表示される。

エラー 1:

CI/CD の「ワークフロー」ページのワークフロー名の下に、次のように表示されます。

`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] を選択して、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。エラーの詳細がページの下部に表示されます。エラーに含まれるトラブルシューティングのヒントに従って、問題を解決してください。
- 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てください。
- 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にネストされていることを確認してください。プロパティをワークフロー定義ファイルにネストする方法については、を参照するかワークフロー定義リファレンス、from 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。アクションの追加
- アスタリスク (*) やその他の特殊文字が適切に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内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 プロパティの大文字と小文字が正しいことを確認してください。ケーシング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区別するにはワークフロー定義リファレンス、を参照するか、またはからリンクされているアクションのマニュアルを参照してください。アクションの追加

- SchemaVersionそのプロパティがワークフロー定義ファイルに存在し、正しいバージョンに設定されていることを確認してください。詳細については、「SchemaVersion」を参照してください。
- Triggersワークフロー定義ファイルのセクションに、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ーでトリガーを選択し、情報が不足しているフィールドを探すか、にある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。
- DependsOnワークフロー定義ファイル内のプロパティが適切に設定され、循環依存が生じていないことを確認してください。詳細については、「他のアクションに依存するアクションの設定」を参照してください。
- Actionsワークフロー定義ファイルのセクションには、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てください。詳細については、「アクション」を参照してください。
- 各アクションには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ー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探すか、from 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。
- 1 つのアクションで定義された変数は必ずエクスポートして、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、「変数をエクスポートして他のアクションでも使用できるようにする。」を参照してください。

「認証情報が見つかりません」エラーと「ExpiredToken」エラーを修正する方法を教えてください。

問題:作業中にチュートリアル:Amazon EKS へのアプリケーションのデプロイ、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一方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。この場合は、aws configureコマンドを実行しないでください。代わりに、AWS 以下の手順に従ってアクセスキーとセッショントークンを更新してください。

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 チュートリアル全文 (codecatalyst-eks-user) で、 AWS 使用しているユーザーのアクセスポータル URL、ユーザー名、パスワードがあることを確認してください。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を完了した時点で、これらの項目は設定されているはずです。

   > **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、[パスワードをリセット] を選択し、[ワンタイムパスワードを生成] を選択します [...] 、パスワードをもう一度リセットすると、画面に情報が表示されます。

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

   - AWS アクセスポータルページがすでに読み込まれている場合は、ページを更新してください。

3. まだサインインしていない場合は、codecatalyst-eks-userのユーザ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、 AWS アカウント codecatalyst-eks-userユーザと権限セットを割り当てたの名前を選択します。
5. 権限セット名 (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。これらは以下のようになります。

   ```
   export AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
   export AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
   export AWS_SESSION_TOKEN="session-token"
   ```

   ... ここで、*######################*。

7. 開発マシンのターミナルプロンプトにコマンドを貼り付け、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。

   認証情報が更新されました。これで AWS CLI、eksctl、、kubectlコマンドが動作するはずです。

「サーバーに接続できません」というエラーを修正する方法を教えてください。

問題:で説明されているチュートリアルの作業中にチュートリアル:Amazon EKS へのアプリケーションのデプロイ、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ています。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、 AWS クラスターがデプロイされているリージョンに置き換えられます。

CodeDeploy ビジュアルエディターにフィールドがないのはなぜですか?

問題:Deploy to Amazon ECS アクションを使用していて、 CodeDeploy CodeDeploy AppSpecワークフローのビジュアルエディタのようなフィールドが表示されない。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 Blue/Green 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、「「Amazon ECS にデプロイ」アクションの追加」を参照してください。
- 選択した Amazon ECS サービスをブルー/グリーンデプロイを実行するように設定します。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 開発者ガイド』の「ブルー/グリーンデプロイメント」を参照してください。 CodeDeploy

IAM 機能のエラーを修正する方法を教えてください。

問題:Deploy AWS CloudFormation スタックアクションを使用していて、Deploy AWS CloudFormation Stack **`##[error] requires capabilities: [capability-name]`** アクションのログに表示される。

考えられる解決方法:以下の手順を実行して、ワークフロー定義ファイルに機能を追加してください。IAM 機能の詳細については、『IAM ユーザーガイド』の「AWS CloudFormation テンプレート内の 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ーを使用して IAM 機能を追加するには

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. ワークフロー図で Deploy AWS CloudFormation Stack アクションを選択します。
8. [設定] タブを選択します。
9. 一番下にある [詳細設定-オプション] を選択します。
10. 「機能」ドロップダウンリストで、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選択します。リストにその機能がない場合は、YAML エディターを使用して追加してください。
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. [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。
13. 新しいワークフローの実行が自動的に開始されない場合は、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るかどうかを確認してください。ワークフローを手動で実行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開始。

YAML

YAML エディターを使用して IAM 機能を追加するには

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. Deploy AWS CloudFormation Stack アクションで、`capabilities`次のようなプロパティを追加します。

```
DeployCloudFormationStack:
  Configuration:
    capabilities: capability-name
```

capability-name は、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てください。複数の機能を列挙するには、コンマを使用し、スペースは入れないでください。詳細については、`capabilities`AWS CloudFormation 「スタックのデプロイ」アクションリファレンスの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. [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。
10. 新しいワークフローの実行が自動的に開始されない場合は、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るかどうかを確認してください。ワークフローを手動で実行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開始。

「npm install」エラーを修正するにはどうすればいいですか?

問題:AWS CDKAWS CDK デプロイアクションまたはブートストラップアクションがエラーで失敗します。`npm install`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) AWS CDK レジストリにアプリの依存関係を格納しているために発生することがあります。

考えられる解決方法:以下の手順に従って、 AWS CDK `cdk.json`レジストリと認証情報を追加してアプリのファイルを更新してください。

開始する前に

1. 認証情報用のシークレットを作成してください。これらのシークレットは、クリアテキストで同等のものを指定する代わりに、`cdk.json`ファイル内で参照します。シークレットを作成するには:

   a. https://codecatalyst.aws/ CodeCatalyst で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. 以下のプロパティで 2 つのシークレットを作成します。

1 つ目のシークレット	第二の秘密
名前: `npmUsername`	名前: `npmAuthToken`
値:*`npm-username`* 。*`npm-username ###### 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.`	値:*`npm-auth-token`* 、プライベート npm *`npm-auth-token`* レジストリ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。 (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シークレットの使用。

2. AWS CDK シークレットを環境変数として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. 以下のプロパティを持つ 2 つの変数を追加します。

1 つ目の変数	2 番目の変数
名前: NPMUSER	名前: NPMTOKEN
値: ${Secrets.npmUsername}	値: ${Secrets.npmAuthToken}

   これで、シークレットへの参照を含む変数が 2 つできました。

ワークフロー定義ファイルの YAML コードは以下のようになるはずです。

> **Note**
>
> AWS CDK 次のコードサンプルはブートストラップアクションからのもので、AWS CDK デプロイアクションも同様です。

```
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、NPMUSERNPMTOKENcdk.jsonファイル内のと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、ファイルを開きます。cdk.json
2. "app":プロパティを探し、*##############################*。

> **Note**
>
> TypeScript 次のサンプルコードは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、コードは似ていますが、同じではありません。

```
{
  "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. *####################*、以下を置き換えてください。

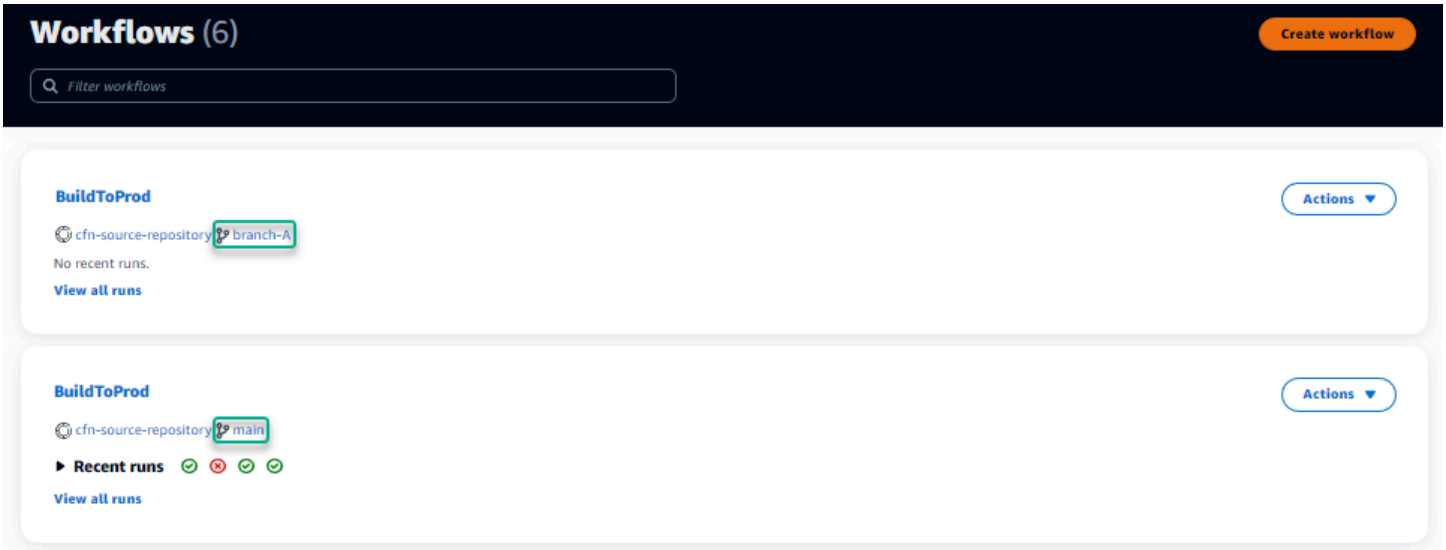
   - *your-registry/folder/CDK-Package/* を、プライベートレジストリ内のプロジェクト依存関係へのパスに置き換えます。 AWS CDK
   - *hello-cdk.ts|.js* とエントリポイントファイルの名前。使用している言語によっては、.ts (TypeScript) または .js () ファイルの場合があります。JavaScript

   > **Note**
   >
   > このアクションは、***NPMUSER ### 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るかどうかを確認します。アクションを手動で実行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開始。

複数のワークフローに同じ名前が付いているのはなぜですか?

ワークフローはブランチごと、リポジトリごとに保存されます。2 つの異なるワークフローが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は、ブランチ名を見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、「Amazon ンの支店との連携 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ーに保存できますか?

いいえ、.codecatalyst/workflowsワークフロー定義ファイルはすべてフォルダーに保存する必要があります。複数の論理プロジェクトでmono リポジトリを使用している場合は、.codecatalyst/workflowsすべてのワークフロー定義ファイルをフォルダーに配置し、FilesChangedトリガー内の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、「トリガーの使用」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方法を教えてください。

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、そのアクションには依存関係がなく、他のアクションとparallel 実行されます。

アクションを順番に並べたい場合は、DependsOnフィールドを設定して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や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アクションの設定」を参照してください。

ワークフローが正常に検証されたのに、実行時には失敗するのはなぜでしょうか?

Validateボタンを使用してワークフローを検証したが、それでもワークフローが失敗した場合は、バリデーターの制限が原因である可能性があります。

ワークフロー設定内のシークレット、環境、 CodeCatalyst フリートなどのリソースに関するエラーは、コミット中には登録されません。有効でない参照が使用された場合、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入力ミスなど、アクション設定にエラーがあった場合は、ワークフローの実行時にのみエラーが特定されます。詳細については、「ワークフローの作成、編集、削除」を参照してください。

オートディスカバリーでは、自分のアクションに関するレポートは見つかりません。

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、 CodeCatalystによってレポートが検出されません。

考えられる解決方法:原因はいくつか考えられます。次の 1 つまたは複数の解決策を試してください。

- テストを実行するツールが、 CodeCatalyst 理解できるフォーマットのいずれかで出力を生成するようにしてください。たとえば、pytest CodeCatalyst テストレポートやコードカバレッジレポートを検出できるようにしたい場合は、次の引数を含めてください。

  ```
  --junitxml=test_results.xml --cov-report xml:test_coverage.xml
  ```

  詳細については、「テストレポートタイプ」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てください。たとえば、pytestJUnitXML結果をフォーマットどおりに出力するように設定する場合は、ファイル拡張子がであることを確認してください.xml。詳細については、「テストレポートタイプ」を参照してください。
- IncludePaths特定のフォルダを意図的に除外する場合を除いて、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てください。同様に、ExcludePathsレポートが保存されるはずのディレクトリも除外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構成した場合、そのファイルは自動検出から除外されます。詳細については、「例:レポートの設定」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、オートディスカバリーはレポートを見つけられない場合があります。たとえば、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。

達成基準を設定した後に、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達成基準を設定すると、一部のレポートが達成基準を満たさず、結果としてアクションが失敗します。

考えられる解決策:この問題を解決するには、次の 1 つまたは複数の解決策を試してください。

- IncludePathsExcludePaths関心のないレポートを変更するか、除外してください。
- すべてのレポートが合格するように達成基準を更新してください。たとえば、1 つのレポートがラインカバレッジが 50%、もう 1 つが 70% のレポートが見つかった場合は、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。
- 問題のある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、特定のレポートに異なる達成基準を設定できます。詳細については、「レポートの達成基準の設定」を参照してください。

オートディスカバリーでは必要のない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。たとえば、 CodeCatalyst に格納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を生成します。node_modules

考えられる解決方法:ExcludePaths不要なファイルを除外するように構成を調整できます。たとえばnode_modules、node_modules/**/*除外するには追加します。詳細については、「パスを含める/除外する」を参照してください。

自動検出では、1 つのテストフレームワークについて多数の小さなレポートが生成されます。

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、自動検出によって大量のレポートが生成されることに気付きました。たとえば、Maven Surefire Plugin を使用する場合、オートディスカバリーはテストクラスごとに異なるレポートを生成します。

考えられる解決方法:ご使用のフレームワークでは、出力を 1 つのファイルに集約できる場合があります。たとえば、Maven Surefire Plugin 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約できます。式全体は以下のようになります。

```
mvn test; cd test-package-path/surefire-reports && npx junit-merge -d ./ && rm
 *Test.xml
```

CI/CD 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ない

問題:CI/CD、Workflows ページに表示されるワークフローが、~/.codecatalyst/workflows/ソースリポジトリ内のフォルダーのワークフローと一致しません。次のような不一致が発生する可能性があります。

- ワークフローは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が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、この 2 つは異なります。

この問題は、ワークフローページを更新する時間がない場合や、ワークフロークォータを超えた場合に発生することがあります。

解決方法:

- 待ちます。通常、ソースにコミットしてからワークフローページに変更が表示されるまで、2 ~ 3 秒待たなければなりません。
- 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てください。

> **Note**
>
> ワークフロークォータを超えているかどうかを判断するには、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、照合してください。のワークフローのクォータ CodeCatalystクォータを超えたことを示すエラーメッセージは表示されないため、自分で調査する必要があります。

- スペースあたりの最大ワークフロー数のクォータを超えた場合は、いくつか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てください。テストコミットの例としては、ファイルにスペースを追加することが挙げられます。

- ワークフロー定義ファイルの最大サイズ制限を超えた場合は、ワークフロー定義ファイルを変更して長さを短くしてください。
- 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、テストコミットを数回実行してください。各コミットのワークフローの最大数より少ない数を変更してください。
- 有料階層の請求を有効にして、ワークフロークォータを増やしてください。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のですが、変更をコミットしようとすると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。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。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、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権限があります。

コントリビューターロールを持っているが、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特定のブランチにコードをプッシュすることを禁止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、変更内容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、「ブランチの許可されたアクションをブランチルールで管理」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、「で検索 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
- 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
- ページ間を移動すると、検索結果の数が変わり続ける
- 検索クエリが完了していません

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。検索クエリを絞り込む

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

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、「AWS アカウント スペースの管理」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、「AWS アカウント スペースの管理」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用の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。の開発環境 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、「Amazon のロールを使った作業 CodeCatalyst」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、「ブランチの許可されたアクションをブランチルールで管理」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 経由で開発環境に接続する」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 経由で開発環境に接続する」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 での開発環境の使用。

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

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
- 開発環境のリポジトリ開発ファイルを移動したいです。
- devfile の起動に問題があります。
- devfile のステータスを確認する方法がわかりません。
- 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

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、「ユニバーサル開発ファイルイメージ」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。
- 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。

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

Amazon CodeCatalyst Health Report は、 up-to-the-minute CodeCatalyst 広範囲に影響を及ぼすリソースのパフォーマンスとサービスの可用性に関する通知を集約したリストをユーザーに提供する公開ダッシュボードです。で、どのリソースに問題があり、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、 CodeCatalyst プロジェクト内のスペース管理者権限を持つすべてのユーザーにアラートと電子メール通知を自動的に送信し、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 日間に発生した過去 100 件まで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。インシデントのリストを更新して、最新の更新を確認することもできます。

CodeCatalyst ヘルスレポートを使用する際には、次のようなワークフローが考えられます。

Mateo Jackson は、スペース管理者権限を持つ新進スペースの開発者です。プルリクエストを作成しようとすると、エラーメッセージが表示され続けます。彼は自分のメールをチェックし、自分のスペースに影響を及ぼしているシステム問題に関する詳細な履歴が記載された自動生成されたシステムインシデントメールを受け取ったことに気付きました。 CodeCatalyst [更新を表示] を選択すると、CodeCatalyst ヘルスレポートが開き、システムから報告されたすべてのインシデントを確認できます。一覧からインシデントを選択して詳細情報を調べます。分割画面が開き、前回の更新のタイムスタンプ、履歴、影響を受けた機能、開始時間、インシデントの現在のステータスが表示されます。また、問題は進行中であるが、サービスチームが作業を開始していることもわかります。インシデントの履歴やステータスが更新されるたびに、メールが届きます。メールにアクセスできない場合は、CodeCatalyst トップパネルのベルのアイコンを選択して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

以下の概念を学習することで、 CodeCatalyst ヘルスレポートを理解し、アプリケーション、サービス、リソースの状態を追跡する方法を理解できます。

インシデント

インシデントとは、 CodeCatalyst内部のアプリケーションやリソースに影響を及ぼしているシステムイベントです。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問題の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のステータスです。[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

影響を受ける機能とは、インシデントの影響を受けた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、イシュー、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:

Updated on には、インシデントの最終更新のタイムスタンプが表示されます。

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用の 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[サインアップして最初のスペースと開発ロールを作成する](#)。
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用の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

次の表では、Amazon のクォータと制限について説明しています。 CodeCatalystの特定の側面に関する追加情報については、 CodeCatalyst 以下のトピックを参照してください。

- のソースリポジトリのクォータ CodeCatalyst
- の ID、許可、アクセスのクォータ CodeCatalyst
- のワークフローのクォータ CodeCatalyst
- の開発環境のクォータ CodeCatalyst
- のプロジェクトのクォータ CodeCatalyst
- でのブループリントのクォータ CodeCatalyst
- 内のスペースのクォータ CodeCatalyst
- の問題のクォータ CodeCatalyst

アカウント内の最大スペース数	5
1 か月間にユーザーが作成できるスペースの最大数	5
1 AWS アカウント つのスペースの最小数	1
1 スペースあたりのアカウント接続の最大数	5,000
スペースの VPC 接続の最大数	100
1 スペース内の最大プロジェクト数	100
1 人のユーザーが所属できるプロジェクトの最大数	1,000
スペースの説明	スペースの説明は省略可能です。指定する場合は、長さが 0 ~ 200 文字でなければなりません。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、および以下の特殊文字を自由に組み合わせることができます。

	? & $ % + = / \ ; : \n \t \r
スペース名	スペース名は全体で一意でなければなりません CodeCatalyst。削除し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 文字でなければなりません。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、ダッシュを自由に組み合わせることができます。次の文字は使用できません。 ! ? @ # $ % ^ & * ( ) + = { } [ ] | ⁄⁄ > < ~ ` ' " ; :

Amazon 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まとめたものです CodeCatalyst。

変更	説明	日付
新コンテンツ:タスクを使用して問題をより小さな目標に分解します	Issue 内のタスクの開始をサポートするコンテンツを追加しました。タスクを課題に追加して、その課題の作業をさらに細分化、整理、追跡できます。	2024 年 4 月 4 日
更新された内容: GitHub でのアクションの制限事項 CodeCatalyst	GitHubアクションが古いランタイム環境の Docker GitHub でのアクションの制限事項 CodeCatalyst イメージで実行されることを示すようにトピックを更新しました。	2024 年 4 月 2 日
更新された内容: アーティファクトによる作業	と 2 アーティファクトによる作業 つの新しいサブトピックを含むようにトピックを更新しました。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? 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
新しいコンテンツ:「AWS CDK デプロイ」アクションリファレンス	`CloudAssemblyRootPath` に新しいプロパティを追加しました「AWS CDK デプロイ」アクションリファレンス。	2024 年 4 月 1 日
更新された内容: ランタイム環境の Docker イメージの操作	2024 年 3 ランタイム環境の Docker イメージの操作 月の新	2024 年 3 月 26 日

	しいランタイム環境イメージに関する情報を含むようにトピックを更新しました。	
内容を更新しました:ロールの操作	ロール権限情報を 1 つのテーブルに統合しました。各ロールで使用できる権限この表は新しいトピックです。	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	CodeCatalyst サインインしているユーザーの各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: プルとブランチを持つトリガー	プルリクエストトリガーの例を追加しました。トリガーの使用トピック全体に小さな修正を加えました。	2024 年 3 月 11 日
内容の更新:ロールを使った作業	ロールのドキュメントを更新し、環境を作成、削除、表示する権限を追加しました。	2024 年 3 月 4 日
コンテンツを更新しました:チュートリアル:ジェネレーティブ AI 機能の使用	課題を作成して 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: GitHub ワークフローが失敗した際のプルリクエストのマージをブロックする	ワークフローが失敗した場合に、 GitHub プルリクエストとブランチ保護ルールとのマージをブロック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 3 月 4 日
新しいコンテンツ:課題、コンポーネント	カスタムブループリント開発者として課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 アクション CodeCatalyst ラボアクションのリストを含むようにトピックを更新しました。	2024 年 2 月 21 日
コンテンツを更新しました:プルリクエストの処理	承認ルールやプルリクエストをマージするためのオーバーライド要件を含む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 2 月 15 日
コンテンツの更新:プルリクエストのマージ	プルリクエストのドキュメントが追加され、必要なレビュアーからまだ承認を受けていない、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オーバーライドに関する情報が含まれるようになりました。	2024 年 2 月 15 日

新コンテンツ: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
コンテンツが更新されました。ロールを使った作業	ロールのドキュメントを更新し、承認ルールとプルリクエストを操作するための権限を追加しました。	2024 年 2 月 14 日
更新された内容: ######### ################## ##############	################## ################## ##セクションを更新して、トラブルシューティングのヒントをさらに追加しました。	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	ワークフローのステータスを説明するセクションを追加しました。	2024 年 2 月 9 日
更新内容:のワークフローのクォータ CodeCatalyst	「ワークフローごとのアクションの最大数」と「 AWS アカウント スペースあたりの割り当てに関連する環境の最大数」のワークフローのクォータ CodeCatalyst でトピックを更新しました。	2024 年 2 月 7 日
更新された内容: 環境を作成する	1 つの環境で最大 1 環境を作成する つのアカウント接続を使用できることを示すように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リポジトリ GitHub	GitHub 公開されているリポジトリに新しいコンテンツを追加しました。	2024 年 1 月 10 日

内容の更新:npm を次のように構成しました。 CodeCatalyst	npm with を使用する際の一般的な設定手順を更新し CodeCatalyst、オプションに関する説明をわかりやすくしました。`always-auth=true`	2024 年 1 月 5 日
内容の更新:プルリクエストの処理	のジェネレーティブ AI 機能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 CodeCatalyst。	2023 年 11 月 28 日
コンテンツの更新:課題の作成	のジェネレーティブ AI 機能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 CodeCatalyst。	2023 年 11 月 28 日
新しいコンテンツ:チュートリアル:ジェネレーティブ AI 機能の使用	Amazon CodeCatalyst のジェネレーティブ 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。	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 CodeCatalyst 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。	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	チュートリアルを更新し、修正とトラブルシューティング情報を追加しました。	2023年11月22日
更新された内容: トリガーの使用	プルリクエストトリガーに関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 CodeCatalyst フェデレーションをサポートするスペースの設定と管理に関する情報へのリンクを追加しました。「セットアップ 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
内容の更新:ロールの操作	ロールのドキュメントを更新し、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権限を追加しました。	2023 年 11 月 16 日
内容を更新しました:プルリクエストの処理	プルリクエスト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
コンテンツの更新:のクォータ CodeCatalyst	スペースクォータの VPC のクォータ CodeCatalyst 接続の最大数に関する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:VPC 接続での開発環境の使用	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:スペースとプロジェクトのチーム管理 CodeCatalyst	スペースのあるチームの使用に関する情報を追加しました。「チーム管理」および「プロジェクトのチーム管理」を参照してください。	2023 年 11 月 16 日

新しいコンテンツ:スペース内のブループリントとワークフロー用のマシンリソースの管理	スペースを使ったマシンリソースの使用に関する情報を追加しました。マシンリソースの管理 を参照してください。	2023 年 11 月 16 日
新しいコンテンツ:プロジェクト内のブループリントとワークフロー用のマシンリソースの管理 CodeCatalyst	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。マシンリソースの管理 を参照してください。	2023 年 11 月 16 日
新しいコンテンツ:VPC 接続と環境の関連付け	VPC 接続をワークフローで使用できる環境に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	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」アクションリファレンス	`CdkCliVersion` 「AWS CDK デプロイ」アクションリファレンスとに新しいプロパティを追加しましたAWS CDK 「ブートストラップ」アクションリファレンス。	2023 年 11 月 14 日
更新内容:ロールの操作	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権限を追加しました。	2023 年 11 月 13 日
内容の更新: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のトピックが更新され、ブランチルールの操作に関する情報が含まれるようになりました。	2023 年 11 月 13 日

更新された内容: ビルドとテストアクションのリファレンス	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	ブランチに関するドキュメントが追加され、ソースリポジトリ内のブランチのルールを表示したり、ブランチルールを作成および管理したりする方法に関する情報が掲載されました。	2023 年 11 月 13 日
内容の更新:プルリクエストの処理	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: ファイルキャッシュの操作	ファイルキャッシュの制限を含む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 へのアプリケーションのデプロイ	EKS アプリデプロイブループリント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:内のパッケージ CodeCatalyst	でのパッケージの使用に関する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:ロールの操作	パワーユーザー、制限付きアクセス、 CodeCatalystレビュー担当者、読み取り専用の 4 つの新しい役割のドキュメントを更新しました。	2023 年 11 月 1 日

更新された内容: GitHub アクション出力パラメータの操作	GITHUB_OUTPUT set-output コマンドの代わりに環境ファイルを使用するように例を更新しました。出力パラメータの設定には、 GitHub 環境ファイルを使用することが推奨されています。	2023 年 10 月 24 日
新しいコンテンツ:トリガーの使用	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 クラスターへのデプロイ」アクションリファレンス	CodeCatalystWorkflowDevelopmentRole-*spaceName* 「Kubernetes クラスターへのデプロイ」アクションリファレンスチュートリアル:Amazon EKS へのアプリケーションのデプロイロールの使用に関する情報がおよびトピックに追加されました。	2023 年 9 月 22 日

内容の更新:CodeCatalystWorkflowDevelopmentRole-*spaceName* 新しいロール名とロールのポリシー	開発者ロール名をに変更する手順とロールの説明がに更新されましたCodeCatalystWorkflowDevelopmentRole-*spaceName* 。`AdministratorAccess` AWS 開発者ロールは管理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「サインアップして最初のスペースと開発ロールを作成する」を参照してください。	2023 年 9 月 20 日
更新された内容: 変数の操作	ユーザー定義変数と定義済み変数という 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、変数の操作このセクションは読みやすく、理解しやすくなるはずです。	2023 年 9 月 19 日
内容の更新:コミットの操作	表示される情報の変更を反映し、複数の親を持つ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:トリガーの使用	トリガーの使用次の例をトピックに追加しました。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

コンテンツを更新しました:プルリクエストの処理	プルリクエスト作成時のソースブランチと宛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
新コンテンツ:デフォルトブランチの表示と変更	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 クラスターへのデプロイ」アクションリファレンス	Helm と Kustomize Manifests に関する注記をのプロパティの説明に追加しました。「Kubernetes クラスターへのデプロイ」アクションリファレンス	2023 年 8 月 15 日
新しいコンテンツ:課題添付ファイルの管理	課題の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: トリガーの使用	ワークフロートリガーに関するドキュメントを改善および拡張しました。	2023 年 8 月 11 日
新しいコンテンツ:ロール権限のトラブルシューティング	Amazon CodeGuru 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権限の更新に関する情報を追加しました。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:「ワークフローは無効です」というメッセージを修正する方法を教えてください。	次のトラブルシューティングトピックが追加されました。「ワークフローは無効です」というメッセージを修正する方法を教えてください。	2023 年 8 月 11 日
新しいコンテンツ:「Kubernetes クラスターへのデプロイ」アクションの追加	Kubernetes へのデプロイクラスターアクションに関するドキュメントを追加しました。詳細については、「「Kubernetes クラスターへのデプロイ」アクションの追加」および「チュートリアル:Amazon EKS へのアプリケーションのデプロイ」を参照してください。	2023 年 7 月 27 日
スペースの管理イベントの記録方法に関する更新 CodeCatalyst	CodeCatalyst AWS CloudTrail との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ついての情報を追加しました。 list-event-logs スペース内のすべてのイベントをコマンドで表示する方法についての情報を追加しました。Amazon でのモニタリング CodeCatalyst を参照してください。	2023 年 7 月 20 日

更新された内容: トリガーの使用	GitHub プルリクエストのトリガーが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説明するようにドキュメントを更新しました。以前は、 CodeCatalyst プルリクエストトリガーは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: ワークフロー定義リファレンス	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
更新された内容: のワークフローのクォータ CodeCatalyst	のワークフローのクォータ CodeCatalystトピックを「アクションで実行できる最大時間」クォータで更新しました。	2023 年 6 月 27 日
更新内容:データ保護	データ複製に関する追加情報を含む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:アクションバージョンでの作業	アクションバージョンでの作業トピックを追加しました。	2023 年 6 月 21 日
更新された内容: 環境を使用する	どのアクションが環境をサポートしていますか？セクションを明確化しました。	2023 年 6 月 14 日
内容の更新:問題のドキュメントを再編しました。	ほとんどの課題ドキュメントを再編成して、ドキュメントセット全体とユーザーフローとの整合性を高めました。	2023 年 5 月 31 日

コンテンツの更新:課題ビュースイッチャー	更新された課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
コンテンツが更新されました:通知の管理	通知に関するドキュメントを更新し、個人用 Slack 通知の設定に関する情報を含めました。	2023 年 5 月 30 日
内容を更新しました:通知の管理	通知に関するドキュメントを更新し、個人用 Slack 通知の設定に関する情報を含めました。	2023 年 5 月 30 日
新コンテンツ: CodeCatalyst トラストモデル	CodeCatalyst 接続環境でのサービスロールの引き継ぎを可能にするトラスト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。の定義済み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。 CodeCatalystCodeCatalyst トラストモデル を参照してください。	2023 年 5 月 20 日
更新された内容: ソースの操作	の手順を簡略化しました。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: のワークフローのクォータ CodeCatalyst	のワークフローのクォータ CodeCatalystトピックを「出力変数値の最大長」クォータで更新しました。	2023 年 5 月 10 日

新しいコンテンツ:アーティファクトによる作業	例:1 つのアーティファクト内のファイルを参照すると 2 つの例を追加しました例:アーティファクト内のファイルがある場合の参照 WorkflowSource 。	2023 年 5 月 10 日
コンテンツの更新:プルリクエストの処理	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メールプリファレンスの設定に関する情報を追加しました。	2023 年 4 月 21 日
更新内容:通知の管理	通知に関するドキュメントを更新し、プルリクエストイベントのメールプリファレンスの設定に関する情報を追加しました。	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および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:デプロイメントターゲットの削除	デプロイメント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 ラボの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アップデート	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。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する 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アップデート	スペース管理者として開発環境を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
更新内容:問題の検索と表示	問題の検索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: コンピュートでの作業	Amazon Linux 2 に Arm64 アーキテクチャのサポートを追加しました。	2023 年 4 月 19 日
新しいコンテンツ:グループ内での課題の移動	ボードビューとすべての課題ビューで、グループ内の課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: のワークフローのクォータ CodeCatalyst	のワークフローのクォータ CodeCatalystクォータが不足しているトピックが更新され、1 つのアクションの出力変数の最大合計サイズが (2 KB から) 120 KB に更新されました。	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を表示する	アクションのソースコードを表示する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:テストケースの再試行	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を付けるためのセクションを追加しました 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」アクションリファレンス	`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: CodeCatalyst GitHub リンクされたリポジトリを使ってプロジェクトを作成する	に、Amazon でのプロジェクトの作成 CodeCatalyst GitHub リポジトリをリンクして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が記載された新しいセクションを追加しました。	2023 年 4 月 5 日

内容の更新:通知の取り扱い	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 アクション開発キットガイドの初版。	2023 年 3 月 31 日
コンテンツの更新:Amazon のスペース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て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 へのアプリケーションのデプロイ	ステップ 1: AWS ユーザーを設定し、 AWS CloudShell AWS IAM Identity Center の代わりにユーザーを作成する方法を説明するように変更されました。 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
更新された内容:ロールの操作	スペース管理者、プロジェクト管理者、コントリビューターの各ロールのドキュメントを更新し、Issue をプルリクエストにリンクする権限を追加しました。	2023 年 3 月 13 日
内容を更新しました:プルリクエストの処理	プルリクエストのドキュメントを更新し、Issue をプルリクエストに関連付ける方法に関する情報が含まれるようになりました。	2023 年 3 月 13 日

コンテンツを更新しました:課題への取り組み	Issue のドキュメントを更新し、Issue をプルリクエストにリンクする方法に関する情報が含まれるようになりました。	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	新しい集約型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エラーがあります」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、編集、削除	新しい UI を反映するように説明を更新しました。	2023 年 3 月 3 日
新しいコンテンツ:を使用する universal-test-runner	を使用する universal-test-runnerトピックを追加しました。	2023 年 3 月 3 日
更新された内容: のワークフローによるビルド、テスト、デプロイ CodeCatalyst	ワークフロー概要ページの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:コミットインごとのコード品質とデプロイステータスの表示 CodeCatalyst	コミットごとのコード品質とデプロイ状況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:"BranchName" と "CommitId" 変数	`BranchName` 新しい定義済み変数が追加されました。	2023 年 2 月 16 日

更新内容: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でメンバーロールの変更、メンバーの招待、および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
コンテンツの更新:Amazon CodeCatalyst コンソールの PAT 管理の手順を追加しました	コンソールで PAT を表示、作成、削除するステップを追加しました。	2023 年 2 月 15 日
更新された内容: ランタイム環境の Docker イメージの操作	デフォルト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: アーティファクトによる作業	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: ソースの操作	ソース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」アクションリファレンス	`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
コンテンツの更新:プルリクエストの更新	プルリクエストの必須または任意のレビュア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:ファイルキャッシュの操作	ワークフローにファイルキャッシュ用のページを追加しました。	2022 年 12 月 20 日
更新内容:プルリクエストの処理	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含めました。	2022 年 12 月 16 日

新しいコンテンツ:「AWS CDK デプロイ」アクションリファレンス	`CdkRootPath` 新しい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:アクション間での計算の共有	アクション間での計算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: アーティファクトによる作業	入力アーティファクトの指定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」アクションリファレンス	GitHub アクション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
コンテンツが更新されました: のプロジェクトのクォータ CodeCatalyst	ドキュメントを更新し、1 スペースあたり最大 100 件のプロジェクトを追加しました。	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

AWS 最新の用語については、『リファレンス』AWS の用語集を参照してください。AWS の用語集