aws

ユーザーガイド

AWS Proton

AWS Proton: ユーザーガイド

Table of Contents

サポート終了通知: 2026 年 10 月 7 日に、 AWS はサポートを終了します AWS Proton。2026 年 10 月 7 日以降、 AWS Proton コンソールまたは AWS Proton リソースにアクセスできなくなります。デプロイされたインフラストラクチャはそのまま残ります。詳細については、AWS Proton 「サービス廃止と移行ガイド」を参照してください。

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

とは AWS Proton

AWS Proton は、次のとおりです。

- サーバーレスおよびコンテナベースのアプリケーションの Infrastructure as Code のプロビジョニングとデプロイの自動化

  この AWS Proton サービスは、2 本柱の自動化フレームワークです。管理者としてのあなたは、サーバーレスアプリケーションとコンテナベースアプリケーション向けの標準化されたインフラストラクチャとデプロイツールを定義するバージョン付きサービステンプレートを作成します。アプリケーション開発者としてのあなたは、使用可能なサービステンプレートのいずれかを選択して開発したアプリケーションやサービスのデプロイを自動化することができます。

  AWS Proton は、古いテンプレートバージョンを使用している既存のサービスインスタンスをすべて識別します。管理者は、ワンクリックでアップグレード AWS Proton をリクエストできます。
- インフラストラクチャの標準化

  プラットフォームチームは、 AWS Proton およびバージョニングされたインフラストラクチャをコードテンプレートとして使用できます。チームは、それらを使用して、アーキテクチャ、インフラストラクチャリソース、および CI/CD ソフトウェアデプロイパイプラインを含む標準アプリケーションスタックを定義して管理します。
- CI/CD と統合されたデプロイ

  開発者が AWS Proton セルフサービスインターフェイスを使用してサービステンプレートを選択すると、コードデプロイ用の標準化されたアプリケーションスタック定義が選択されます。 はリソース AWS Proton を自動的にプロビジョニングし、CI/CD パイプラインを設定し、定義されたインフラストラクチャにコードをデプロイします。

AWS Proton プラットフォームチーム向け

管理者、またはプラットフォームチームのメンバーは、Infrastructure as Code を含む環境テンプレートおよびサービステンプレートを作成します。環境テンプレートは、複数のアプリケーションまたはリソースで使用される共有インフラストラクチャを定義します。サービステンプレートは、環境内の単一のアプリケーションまたはマイクロサービスをデプロイおよび維持するために必要なインフラストラクチャのタイプを定義します。 AWS Proton サービスはサービステンプレートのインスタンス化であり、通常は複数のサービスインスタンスとパイプラインが含まれます。 AWS Proton サービスインスタンスは、特定の環境内のサービステンプレートをインスタンス化したものです。自

分またはチームのメンバーは、所与のサービステンプレートと互換性がある環境テンプレートを指定できます。テンプレートの詳細については、「AWS Proton テンプレート」を参照してください。

次のインフラストラクチャをコードプロバイダーとして使用できます AWS Proton。

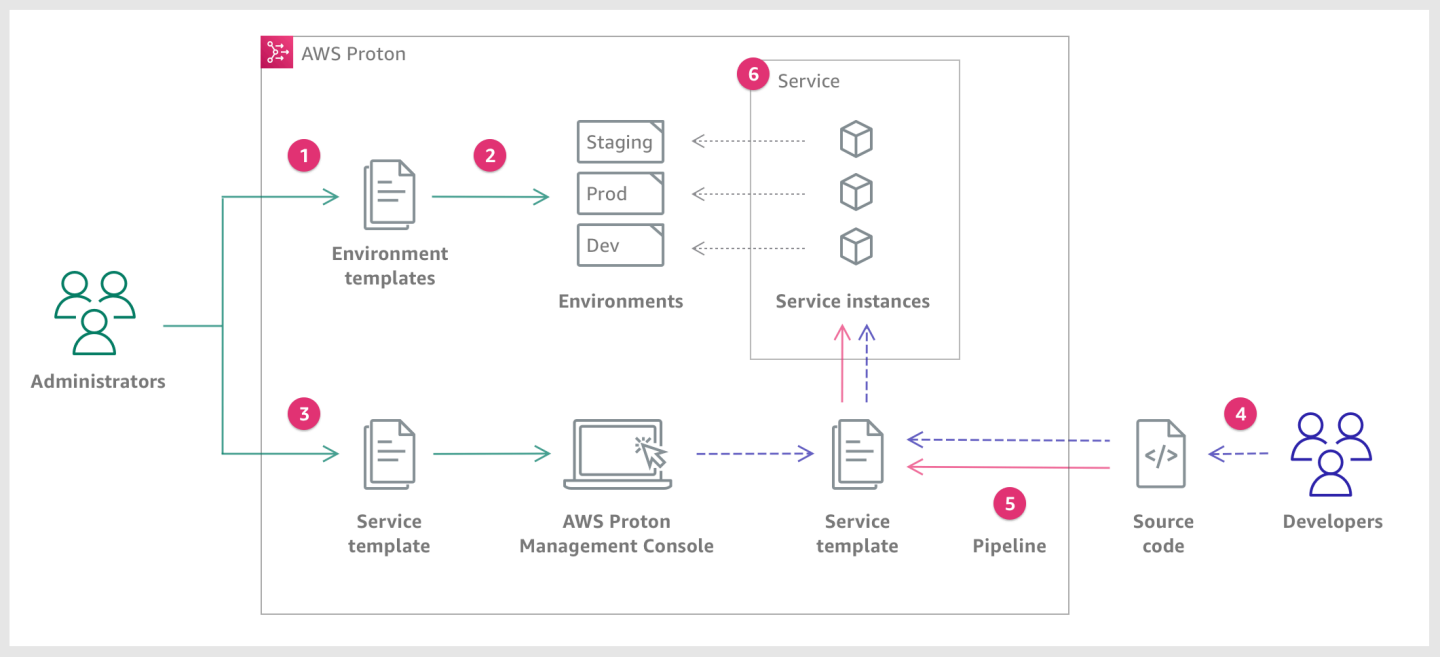
- CloudFormation
- Terraform

AWS Proton デベロッパー向け

アプリケーション開発者は、 AWS Proton が サービスインスタンスにアプリケーションをデプロイして管理するサービスを作成するために使用する標準化されたサービステンプレートを選択します。 AWS Proton サービスは、サービステンプレートのインスタンス化したものであり、通常そこにはサービスインスタンスとパイプラインが含まれます。

AWS Proton ワークフロー

次の図は、前の段落で説明した主な AWS Proton 概念を視覚化したものです。また、シンプルな AWS Proton ワークフローを構成するものの概要も示します。



1 管理者は、共有リソース AWS Protonを定義する環境テンプレートを作成して に登録します。

2

AWS

Proton は、環境テンプレートに基づいて 1 つ以上の環境をデプロイします。

3

管理者は、関連するインフラストラクチャ、モニタリング AWS Proton、CI/CD リソース、および互換性のある環境テンプレートを定義するサービステンプレートを作成して に登録します。

4

開発者としてのあなたは、登録済みサービステンプレートを選択し、ソースコードリポジトリのリンクを提供します。

5

AWS Proton は、サービスインスタンスの CI/CD パイプラインを使用してサービスをプロビジョニングします。

6

AWS Proton は、選択したサービステンプレートで定義されているソースコードを実行しているサービスおよびサービスインスタンスをプロビジョニングおよび管理します。 サービスインスタンスは、パイプラインの単一段階 (たとえば、Prod) について環境内で選択したサービステンプレートをインスタンス化したものです。

AWS Proton サービスの廃止と移行ガイド

AWS は中止を決定し AWS Proton、サポートは 2026 年 10 月 7 日に終了します。新規のお客様は 2025 年 10 月 7 日以降にサインアップすることはできませんが、既存のお客様は 2026 年 10 月 7 日までサービスを引き続き使用できます。

廃止までのサービスステータス

2026 年 10 月 7 日まで、既存の AWS Proton お客様は引き続きこのサービスを通常どおり使用できます。この期間中、 AWS は以下を行います。

1. セキュリティパッチと重要なバグ修正を提供する
2. サービスの可用性とパフォーマンスを維持する
3. AWS サポート チャネルを通じてサポートを提供し続ける
4. サービスに新機能を追加しない

重要な移行情報

AWS Proton は主にインフラストラクチャをデプロイするための CI/CD ツールです。 AWS Proton が廃止されると、デプロイされた CloudFormation スタックとそれらが管理するリソースはそのまま残り、引き続き機能します。非推奨は、デプロイされたインフラストラクチャではなく、配信パイプラインと AWS Proton サービス自体にのみ影響します。

代替ソリューション

インフラストラクチャ AWS Proton をコードおよび CI/CD 機能として維持するのに役立つ の代替案をいくつか特定しました。

CloudFormation Git 同期

最適: GitOps ワークフローを必要とする CloudFormation を使用するチーム

Git 同期を使用すると、プラットフォームチームは開発チームが分岐できる git リポジトリで CloudFormation テンプレートをモデル化できます。開発者はパラメータファイルを更新し、フォークされたリポジトリに変更をプッシュし、Git 同期はスタックを更新します。

主な利点:

1. と同様のデベロッパーエクスペリエンス AWS Proton
2. 既存の CloudFormation の知識を活用する
3. プラットフォームチームとデベロッパーチーム間の明確な分離

制限:

1. 環境の概念がない
2. 高度なパイプライン機能なし
3. 他の Git プロバイダーでは利用できない GitHub の機能に依存する

詳細: Git 同期

Harmonix オン AWS

最適: 包括的な内部開発者ポータルを必要とする企業

Harmonix は Backstage.io に基づく AWS Partner ソリューションであり、チームが Proton と同様のテンプレート、環境、サービスを作成できるようにする AWS プラグインを提供します。

主な利点:

1. と同様の機能 AWS Proton
2. 一般的な Backstage フレームワーク上に構築
3. 完全なデベロッパーポータルエクスペリエンス

制限:

1. AWS のサービス チームが管理していない
2. カスタマイズが必要な可能性のあるリファレンス実装

詳細については、https://harmonixonaws.io/ を参照してください。

AWS CodePipeline および AWS CodeBuild

最適: 最大限の柔軟性と制御を必要とするチーム

AWS 基本的な CI/CD サービスを使用して、柔軟性と制御性に優れた AWS Proton 機能をレプリケートします。

主な利点:

1. 最大限の柔軟性
2. AWS サービスとの深い統合
3. アクティブなメンテナンスと新機能

制限:

1. より多くの実装作業が必要
2. out-of-box開発者セルフサービスの削減

詳細はこちら:

とは AWS CodePipeline

とは AWS CodeBuild

GitHub Actions

最適: シンプルさを求める GitHub を使用する小規模なチーム

>主な利点

1. GitHub リポジトリとの簡単な統合
2. GitHub ユーザーのシンプルなセットアップ
3. 再利用可能なアクションの大規模なマーケットプレイス

制限:

1. GitHub エコシステムに関連
2. プラットフォームチームコントロールにはさらに多くの作業が必要になる場合があります

詳細はこちら:

GitHub Actions ドキュメント

CI/CD の例: GitHub Actions との統合 – ウェブアプリを Amazon EC2 にデプロイするための CI/CD パイプライン

移行ガイダンス

移行プロセスは、実装と選択した代替案によって異なります。一般的なステップ:

1. Proton リソースをインベントリします。
2. 代替ソリューションを選択します。
3. テンプレートデータを抽出します。
4. 選択した代替手段を実装します。
5. 本番ワークロードの移行:

特定の移行サポートについては、 AWS サポート または アカウントチームにお問い合わせください。

よくある質問

Q: AWS が中止するのはなぜですか AWS Proton? A: 他の AWS および AWS Partner ソリューションを通じて、Infrastructure as Code ポリシーの適用に関するお客様のニーズを満たすより良い機会を特定しました。

Q: 廃止日以降も既存のインフラストラクチャは引き続き機能しますか? A: はい。 は主に CI/CD ツール AWS Proton です。デプロイされた CloudFormation スタックとそれらが管理するリソースはそのまま残り、引き続き機能します。非推奨は、デプロイされたインフラストラクチャではなく、配信パイプラインのみに影響します。

Q: 移行に関するサポートを受けるにはどうすればよいですか? A: AWS サポート は移行を支援できます。 に問い合わせるかAWS サポート、 AWS アカウント マネージャーに連絡してサポートを依頼してください。

Q: どの選択肢を選択すればよいですか? A: 最適な代替方法は、特定のユースケースによって異なります。

1. シンプルな GitOps ワークフローの場合: CloudFormation Git Sync
2. デベロッパーポータルを必要とする企業の場合: Harmonix On AWS
3. 最大限の柔軟性を得るには: AWS CodePipeline および AWS CodeBuild
4. GitHub に既に存在するチームの場合: GitHub Actions

Q: 2026 年 10 月 7 日までに移行しない場合どうなりますか? A: アクセスできなくなります AWS Proton。既存のインフラストラクチャは引き続き機能しますが、 AWS Proton を使用して管理または更新することはできません。

Q: データはどのくらいの期間保持されますか? A: 2026 年 10 月 7 日まで。この日付を過ぎると、すべてのデータが削除されます。

他にご質問がある場合は、 にお問い合わせください AWS サポート。

セットアップ

このセクションのタスクを完了すると、サービスおよび環境テンプレートを作成して登録できます。を使用して環境とサービスをデプロイするには、これらが必要です AWS Proton。

> **Note**
>
> 追加費用は AWS Proton かかりません。サービスおよび環境テンプレートは無料で作成、登録、および維持管理ができます。また、ストレージ、セキュリティ、デプロイなど、独自のオペレーションを自己管理 AWS Proton することもできます。の使用中に発生する費用は、次のとおり AWS Proton です。
>
> - デプロイと保守を指示した AWS クラウド リソース AWS Proton のデプロイと使用のコスト。
> - コードリポジトリ AWS CodeStar への接続を維持するコスト。
> - AWS Protonへの入力を提供する必要がある Amazon S3 バケットの維持管理コスト。あなたの the section called “テンプレートバンドル” に Git リポジトリを使用する the section called “テンプレートの同期設定” に切り替えれば、これらのコスト発生を回避できます。

トピック

IAM によるセットアップ

にサインアップすると AWS、 AWS アカウント を含む のすべてのサービスに が自動的にサインアップされます AWS AWS Proton。実際に使用した分のサービスとリソースについてのみ請求されます。

> **Note**
>
> 1 つのチームは、管理者と開発者を含めて、全員が同じアカウントに属する必要があります。

にサインアップする AWS

がない場合は AWS アカウント、次の手順を実行して作成します。

にサインアップするには AWS アカウント

1. https://portal.aws.amazon.com/billing/signup を開きます。
2. オンラインの手順に従います。

   サインアップ手順の一環として、電話またはテキストメッセージを受け取り、電話キーパッドで検証コードを入力します。

   にサインアップすると AWS アカウント、 AWS アカウントのルートユーザー が作成されます。ルートユーザーには、アカウントのすべての AWS のサービス とリソースへのアクセス権があります。セキュリティベストプラクティスとして、ユーザーに管理アクセス権を割り当て、ルートユーザーアクセスが必要なタスクの実行にはルートユーザーのみを使用するようにしてください。

IAM ユーザーの作成

管理者ユーザーを作成するには、以下のいずれかのオプションを選択します。

管理者を管理する方法を 1 つ選択します	目的	方法	以下の操作も可能
IAM Identity Center 内 (推奨)	短期の認証情報を使用して AWSにアクセスします。 これはセキュリティのベストプラクティスと一致しています。ベストプラクティスの詳細については、IAM	AWS IAM アイデンティティセンター ユーザーガイドの「開始方法」の手順に従います。	AWS Command Line Interface ユーザーガイドの を使用する AWS CLI ように を設定 AWS IAM アイデンティティセンターして、プログラムによるアクセスを設定します。

管理者を管理する方法を 1 つ選択します	目的	方法	以下の操作も可能
	ユーザーガイドの「IAM でのセキュリティのベストプラクティス」を参照してください。		
IAM 内 (非推奨)	長期認証情報を使用して AWSにアクセスする。	IAM ユーザーガイドの「緊急アクセス用の IAM ユーザーを作成する」の手順に従います。	IAM ユーザーガイドの「IAM ユーザーのアクセスキーを管理する」の手順に従って、プログラムによるアクセスを設定します。

AWS Proton サービスロールの設定

AWS Proton ソリューションのさまざまな部分用に作成したい IAM ロールがいくつかあります。IAM コンソールを使用して事前に作成することも、 AWS Proton コンソールを使用して作成することもできます。

環境 AWS Proton ロールを作成して AWS のサービス、 AWS Proton がユーザーに代わって他の AWS CodeBuild CloudFormationやさまざまなコンピューティングおよびストレージサービスに対して API コールを行い、リソースをプロビジョニングできるようにします。環境やそこで実行されるサービスインスタンスがAWSマネージドプロビジョニングを使用する場合は、AWSマネージドプロビジョニングロールが必要です。CodeBuild ロールは、環境またはそのサービスインスタンスがCodeBuild プロビジョニングを使用する場合に必要です。 AWS Proton 環境ロールの詳細については、「」を参照してくださいthe section called “IAM ロール”。環境を作成するときは、 AWS Proton コンソールを使用して、これら 2 つのロールのいずれかの既存のロールを選択するか、管理者権限を持つロールを作成できます。

同様に、CI/CD パイプラインをプロビジョニングするために、 AWS Proton がユーザーに代わって他の のサービスに API コールを実行できるように、パイプライン AWS Proton ロールを作成しま

す。 AWS Proton パイプラインロールの詳細については、「」を参照してくださいthe section called “パイプラインのサービスロール”。CI/CD 設定の詳細については、「the section called “アカウント CI/CD パイプラインの設定”」を参照してください。

> **Note**
>
> AWS Proton テンプレートで定義するリソースがわからないため、コンソールを使用して作成するロールには広範なアクセス許可があり、 AWS Proton パイプラインサービスロールと AWS Proton サービスロールの両方として使用できます。本稼働デプロイでは、 AWS Proton パイプラインサービスロールと AWS Proton 環境サービスロールの両方にカスタマイズされたポリシーを作成して、デプロイする特定のリソースへのアクセス許可の範囲を絞り込むことをお勧めします。これらのロールを作成およびカスタマイズするには、 AWS CLI または IAM を使用します。詳細については、「のサービスロール AWS Proton」および「サービスを作成する」を参照してください。

でのセットアップ AWS Proton

を使用して AWS Proton APIs AWS CLI を実行する場合は、インストールされていることを確認します。まだインストールしていない場合、「のセットアップ AWS CLI」を参照してください。

AWS Proton 特定の設定:

- テンプレートを作成および管理するには
  - テンプレート同期設定で、AWS CodeStar 接続をセットアップします。
  - あるいは、Amazon S3 バケットを設定します。
- インフラストラクチャをプロビジョニングするには
  - セルフマネージド型プロビジョニングでは、AWS CodeStar 接続を設定する必要があります。
- (オプション) パイプラインをプロビジョニングするには
  - AWSマネージドプロビジョニングと CodeBuild ベースのプロビジョニングでは、パイプラインロールを設定します。
  - セルフマネージドプロビジョニングの場合は、パイプラインリポジトリを設定します。

プロビジョニングの方法の詳細については、「the section called “AWSマネージドプロビジョニング”」を参照してください。

Amazon S3 バケットのセットアップ

S3 バケットを設定するには、「ユーザーの最初の S3 バケットを作成する」の手順に従って S3 バケットを設定します。入力を が取得 AWS Proton できるバケット AWS Proton に に配置します。これらの入力はテンプレートバンドルと呼ばれます。詳細については、このガイドで当該セクションを参照してください。

AWS CodeStar 接続のセットアップ

リポジトリ AWS Proton に接続するには、サードパーティーのソースコードリポジトリで新しいコミットが行われたときにパイプラインをアクティブ化する AWS CodeStar 接続を作成します。

AWS Proton は接続を使用して以下を行います。

- ユーザーのリポジトリソースコードで新しいコミットが発生すると、サービスパイプラインがアクティブになります。
- Infrastructure as Code リポジトリについてプルリクエストを作成します。
- テンプレートのリポジトリにコミットがプッシュされるたびに、テンプレートのいずれかを変更する新しいテンプレートマイナーバージョンまたはメジャーバージョンを (バージョンがまだ存在しなければ) 作成します。

CodeConnections を使用して Bitbucket、GitHub、GitHub Enterprise、GitHub Enterprise Server リポジトリに接続できます。詳細については、「 AWS CodePipeline ユーザーガイド」のCodeConnections」を参照してください。

CodeStar 接続をセットアップするには

1. AWS Proton コンソールを開きます。
2. ナビゲーションペインで [Settings (設定)] を選択してから [Repository connections (レポジトリ接続)] を選択すると、[Developer Tools (開発者ツール)] [[Settings (設定)] の [Connections (接続)] ページが表示されます。このページには、接続のリストが表示されます。
3. [Create connection] (接続を作成する) を選択し、指示に従って操作します。

アカウント CI/CD パイプラインの設定

AWS Proton は、アプリケーションコードをサービスインスタンスにデプロイするための CI/CD パイプラインをプロビジョニングできます。パイプラインのプロビジョニングに必要な AWS Proton 設定は、パイプラインに選択したプロビジョニング方法によって異なります。

AWSマネージドプロビジョニングと CodeBuild ベースプロビジョニング — パイプラインロールの設定

AWSマネージドプロビジョニングと CodeBuild プロビジョニングを使用すると、 はパイプラインを AWS Proton プロビジョニングします。したがって、 にはパイプラインをプロビジョニングするためのアクセス許可を提供するサービスロール AWS Proton が必要です。これら 2 つのプロビジョニング方法ではそれぞれ独自のサービスロールを使用します。これらのロールはすべての AWS Proton サービスパイプラインで共有され、アカウント設定で 1 回設定します。

コンソールでパイプラインサービスロールを作成する

1. AWS Proton コンソールを開きます。
2. ナビゲーションペインで、[Settings (設定)] を選択し、[Accounts settings (アカウント設定)] を選択します。
3. [アカウント CI/CD 設定] ページで、[設定] を選択します。
4. 次のいずれかを行います。
   - でパイプラインサービスロール AWS Proton を作成するには

     [パイプラインの AWSマネージドプロビジョニングを有効にするには] - [AWSマネージドプロビジョニングパイプラインロール] セクションの [アカウント設定の設定] ページ:

     a. [新しいサービスロール] を選択します。
     b. ロールの名前を入力します (例: **`myProtonPipelineServiceRole`**)。
     c. アカウントの管理者権限を持つ AWS Proton ロールを作成することに同意するには、チェックボックスをオンにします。

     [CodeBuild ベースのパイプラインのプロビジョニングを有効にするには] - [アカウント設定の設定] ページの [CodeBuild パイプラインロール] セクションで [既存のサービスロール] を選択し、[CloudFormation パイプラインロール] セクションで作成したサービスロールを選択します。CloudFormation パイプラインロールを割り当てていない場合は、前の 3 つのステップを繰り返して新しいサービスロールを作成します。

- 既存のパイプラインサービスロールを選択する

  [AWSパイプラインのマネージドプロビジョニングを有効にするには] - [アカウント設定] ページの [AWSマネージドプロビジョニングパイプラインロール] セクションで、[既存のサービスロール] を選択し、あなたの AWS アカウント内のサービスロールを選択します。

  [パイプラインの CodeBuild プロビジョニングを有効にするには] アカウント設定の設定ページで、CodeBuild パイプラインプロビジョニングロールセクションで、既存のサービスロールを選択し、 AWS アカウントのサービスロールを選択します。

5. [Save changes] (変更の保存) をクリックします。

   新しいパイプラインサービスロールが [Account settings (アカウント設定)] ページに表示されます。

セルフマネージドプロビジョニング — パイプラインリポジトリを設定する

セルフマネージドプロビジョニングでは、 は設定したプロビジョニングリポジトリにプルリクエスト (PR) AWS Proton を送信し、自動化コードがパイプラインのプロビジョニングを担当します。したがって、パイプラインをプロビジョニングするためにサービスロール AWS Proton は必要ありません。代わりに、登録済みのプロビジョニングリポジトリが必要です。リポジトリ内のあなたの自動化コードは、パイプラインをプロビジョニングするための権限を与える適切なロールを引き受ける必要があります。

コンソールでパイプラインプロビジョニングリポジトリを登録する

1. 新しいサービス用に CI/CD パイプラインプロビジョニングリポジトリロールを作成していない場合、そのロールを作成します。セルフマネージドプロビジョニングのパイプラインの詳細については、「the section called “セルフマネージド型のプロビジョニング”」を参照してください。
2. ナビゲーションペインで、[Settings (設定)] を選択し、[Account settings (アカウント設定)] を選択します。
3. [アカウント CI/CD 設定] ページで、[設定] を選択します。
4. [Configure account settings] (アカウント設定の構成) ページの [CI/CD pipeline repository] (CI/CD パイプラインリポジトリ) セクションで以下の操作をします。

   a. [新規リポジトリ] を選択し、リポジトリプロバイダーの 1 つを選択します。

   b. [CodeStar connection (CodeStar 接続)] であなたの接続のいずれかを選択します。

> **Note**
>
> 関連するリポジトリプロバイダーアカウントにまだ接続していない場合は、[新しい CodeStar 接続を追加] を選択し、接続作成プロセスを完了してから、[CodeStar 接続] メニューの横にある更新ボタンを選択します。これで、メニューであなたの新しい接続を選択できます。

c. [リポジトリ名] には、あなたのパイプラインプロビジョニングリポジトリを選択します。ドロップダウンメニューには、プロバイダーアカウントのリポジトリのリストが表示されます。

d. [ブランチ名] で、リポジトリのブランチを選択します。

5. [Save changes (変更の保存)] をクリックします。

   あなたのパイプラインリポジトリは [アカウント設定] ページに表示されます。

のセットアップ AWS CLI

を使用して AWS Proton API コールを行う AWS CLI には、最新バージョンの がインストールされていることを確認します AWS CLI。詳細については、AWS Command Line Interface ユーザーガイドのAWS CLIの開始を参照してください。次に、 AWS CLI で の使用を開始するには AWS Proton、「」を参照してくださいthe section called “CLI の開始”。

の開始方法 AWS Proton

開始する前に、 を使用するようにセットアップ AWS Proton し、開始方法の前提条件を満たしていることを確認します。

以下のパスを 1 つ以上選択 AWS Proton して、 の使用を開始します。

- ドキュメントリンク経由でガイド付きのコンソールまたは CLI ワークフローの例に従います。
- ガイド付きのコンソールワークフロー例を参照してください。
- ガイド付きサンプル AWS CLI ワークフローを実行します。

トピック

前提条件

の使用を開始する前に AWS Proton、以下の前提条件を満たしていることを確認してください。詳細については、「セットアップ」を参照してください。

- 管理者権限がある IAM アカウントをお持ちです。詳細については、「IAM によるセットアップ」を参照してください。
- AWS Proton サービスロールがあり、 AWS Proton パイプラインサービスロールがアカウントにアタッチされている。詳細については、「AWS Proton サービスロールの設定」および「のサービスロール AWS Proton」を参照してください。
- AWS CodeStar 接続がある。詳細については、「AWS CodeStar 接続のセットアップ」を参照してください。
- CloudFormation テンプレートの作成と Jinja のパラメータ化に精通しています。詳細については、 CloudFormation 「 ユーザーガイド」および「Jinja ウェブサイト」の「What is CloudFormation?」を参照してください。 https://palletsprojects.com/projects/jinja

- AWS インフラストラクチャサービスに関する実務上の知識がある。
- にログインしています AWS アカウント。

ワークフローの開始方法

サンプルの手順とリンクに従って、テンプレートバンドルの作成、テンプレートの作成と登録、環境およびサービスの作成について学習します。

開始する前に、AWS Proton サービスロールが作成済みであることを確認してください。

サービステンプレートに AWS Proton サービスパイプラインが含まれている場合は、AWS CodeStar 接続とAWS Proton パイプラインサービスロールが作成されていることを確認します。

詳細については、AWS Proton 「サービス API リファレンス」を参照してください。

例; ワークフローの開始方法

1. AWS Proton 入出力の概要については、の AWS Proton 仕組み「」の図を参照してください。
2. 環境バンドルとサービステンプレートバンドルを作成します。
   a. 入力パラメータを識別します。
   b. スキーマファイルを作成します。
   c. infrastructure as code (IaC) ファイルを作成します。
   d. あなたのテンプレートバンドルをまとめるには、マニフェストファイルを作成し、あなたの iaC ファイル、マニフェストファイル、およびスキーマファイルをディレクトリ内に編成します。
   e. テンプレートバンドルにアクセスできるようにします AWS Proton。
3. 環境テンプレートバージョンを作成して に登録します AWS Proton。

   コンソールでテンプレートを作成して登録すると、テンプレートのバージョンが自動的に作成されます。

   を使用してテンプレート AWS CLI を作成および登録する場合:

   a. 環境テンプレートを作成します。
   b. 環境テンプレートのバージョンを作成します。

詳細については、『AWS Proton API リファレンス』の「CreateEnvironmentTemplate」と「CreateEnvironmentTemplateVersion」を参照してください。

4. あなたの環境テンプレートをパブリッシュして使用可能にします。

   詳細については、『AWS Proton API リファレンス』の「UpdateEnvironmentTemplateVersion」を参照してください。

5. 環境を作成するには、パブリッシュ済み環境テンプレートのバージョンを選択し、必要な値を入力します。

   詳細については、『https://docs.aws.amazon.com/proton/latest/APIReference/API_CreateEnvironment.html API リファレンス』の「AWS Proton CreateEnvironment」を参照してください。

6. サービステンプレートバージョンを作成して に登録します AWS Proton。

   コンソールでテンプレートを作成して登録すると、テンプレートのバージョンが自動的に作成されます。

   を使用してテンプレート AWS CLI を作成および登録する場合:

   a. サービステンプレートを作成します。
   b. サービステンプレートのバージョンを作成します。

   詳細については、『AWS Proton API リファレンス』の「CreateServiceTemplate」と「CreateServiceTemplateVersion」を参照してください。

7. あなたのサービステンプレートをパブリッシュして使用可能にします。

   詳細については、『AWS Proton API リファレンス』の「UpdateServiceTemplateVersion」を参照してください。

8. サービスを作成するには、パブリッシュ済みサービステンプレートのバージョンを選択し、必要な値を入力します。

   詳細については、『AWS Proton API リファレンス』の「CreateService」を参照してください。

の開始方法 AWS マネジメントコンソール

の使用を開始する AWS Proton

- 環境テンプレートを作成して表示します。
- 作成した環境テンプレートを使用するサービステンプレートを作成し、表示して、パブリッシュします。
- 環境とサービスを作成します (オプション)。
- サービステンプレート、環境テンプレート、環境、およびサービスを作成してあるた場合、それを削除します。

ステップ 1: AWS Proton コンソールを開く

- AWS Proton コンソールを開きます。

ステップ 2：例題テンプレートの使用準備を整える

1. Github までのコードスター接続情報を作成し、その接続情報に my-proton-connection という名前を付けます。
2. https://github.com/aws-samples/aws-proton-cloudformation-sample-templates に移動します。
3. あなたの Github アカウントにリポジトリのフォークを作成します。

ステップ 3: 環境テンプレートを作成する

ナビゲーションペインで [Environment templates (環境テンプレート)] を選択します。

1. [Environment templates (環境テンプレート)] ページで [Create Environment template (環境テンプレートを作成する)] を選択します。
2. [Create environment template (環境テンプレートを作成する)] ページの [Template options (テンプレートオプション)] セクションで [Create a template for provisioning new environments (新しい環境のプロビジョニング用のテンプレートを作成する)] を選択します。
3. [Template bundle source (テンプレートバンドルソース)]セクションで [Sync template bundle from Git (Git からテンプレートバンドルを同期する)] を選択します。
4. 「テンプレート定義リポジトリ」セクションで、「リンクされている Git リポジトリを選択」を選択します。

5. [リポジトリリスト] から [my-proton-connection] を選択します。
6. [ブランチリスト] から [main] を選択します。
7. [Proton 環境テンプレートの詳細] セクションで。
   a. テンプレート名として **`fargate-env`** を入力します。
   b. 環境テンプレート表示名として **`My Fargate Environment`** を入力します。
   c. (オプション) 環境テンプレートの説明を入力します。
8. (オプション) [Tags (タグ)] セクションで [Add new tag (新しいタグを追加)] を選択し、キーと値を入力してカスタマーマネージドタグを作成します。
9. [Create Environment template (環境テンプレートの作成)] を選択します。

   新しいページが開き、新しい環境テンプレートのステータスと詳細が表示されます。これらの詳細には、 AWS とカスタマーマネージドタグのリストが含まれます。 は、リソースの作成 AWS Proton 時に AWS マネージドタグ AWS Proton を自動的に生成します。詳細については、「AWS Proton リソースとタグ付け」を参照してください。
10. 新しい環境テンプレートのステータスは [Draft (ドラフト)] から始まります。その表示やアクセスができるのは、あなたと `proton:CreateEnvironment` 権限がある他のユーザーです。次の手順に従って、テンプレートを自分以外のユーザーが使用できるようにします。
11. [Template versions (テンプレートのバージョン)] セクションで、先ほど作成したテンプレートのマイナーバージョン (1.0) の左側にあるラジオボタンを選択します。別の方法としては、情報アラートバナーで [Publish (パブリッシュ)] を選択して次のステップをスキップします。
12. [Template versions (テンプレートのバージョン)] セクションで [Publish (パブリッシュ)] を選択します。
13. テンプレートのステータスが [Published (パブリッシュ)] に変わります。最新バージョンのテンプレートなので、これは [Recommended (推奨)] バージョンです。
14. ナビゲーションペインで [Environment templates (環境テンプレート)] を選択します。

    新しいページに環境テンプレートのリストとテンプレートの詳細が表示されます。

ステップ 4: サービステンプレートを作成する

サービステンプレートを作成します。

1. ナビゲーションペインで [Service templates (サービステンプレート)] を選択します。

2. [Service templates (サービステンプレート)] ページで [Create Service template (サービステンプレートを作成する)] を選択します。
3. [Create service template (サービステンプレートの作成)] ページの [Template bundle source (テンプレートバンドルソース)] セクションで [Sync a template bundle from Git (Gitからのテンプレートバンドルと同期)] を選択します。
4. [テンプレート] セクションで、[リンクされた Git リポジトリを選択] を選択します。
5. [リポジトリリスト] から[my-proton-connection] を選択します。
6. [ブランチリスト] から [main] を選択します。
7. [Proton サービステンプレートの詳細] セクションで。

   a. サービステンプレート名として **`backend-fargate-svc`** を入力します。

   b. サービステンプレート表示名として **`My Fargate Service`** を入力します。

   c. (オプション) サービステンプレートの説明を入力します。
8. [Compatible environment templates (互換性のある環境テンプレート)] セクションで以下の操作をします。

   - 環境テンプレート My Fargate Environment の左側にあるチェックボックスをオンにして、新しいサービステンプレートについて互換性のある環境テンプレートを選択します。
9. [Encryption settings (暗号化設定)] については、デフォルトのままにしておきます。
10. [パイプライン定義] セクションで。

    - [このテンプレートには CI/CD パイプラインが含まれています] ボタンは選択したままにしておきます。
11. [Create service template (サービステンプレートの作成)] を選択します。

    これで、 やカスタマーマネージドタグのリストなど、新しいサービステンプレートのステータス AWS と詳細を表示する新しいページが表示されます。
12. 新しいサービステンプレートステータスのステータスは [Draft (ドラフト)] から始まります。アクセスできるのは管理者のみです。サービステンプレートを開発者が使用できるようにするには、次の手順に従います。
13. [Template versions (テンプレートのバージョン)] セクションで、先ほど作成したテンプレートのマイナーバージョン (1.0) の左側にあるラジオボタンを選択します。別の方法としては、情報アラートバナーで [Publish (パブリッシュ)] を選択して次のステップをスキップします。
14. [Template versions (テンプレートのバージョン)] セクションで [Publish (パブリッシュ)] を選択します。

15. テンプレートのステータスが [Published (パブリッシュ)] に変わります。

    サービステンプレートの最初のマイナーバージョンがパブリッシュされ、そのテンプレートは開発者が使用できます。最新バージョンのテンプレートなので、これは [Recommended (推奨)] バージョンです。

16. ナビゲーションペインで [Service templates (サービステンプレート)] を選択します。

    新しいページには、サービステンプレートと詳細のリストが表示されます。

ステップ 5: 環境を作成する

ナビゲーションペインで [Environment (環境)] を選択します。

1. [Create environment (環境の作成)] を選択します。
2. [Choose an environment template (環境テンプレートを選択する)] ページで、先ほど作成したテンプレートを選択します。My Fargate Environment という名前が付いています。次いで、[Configure (設定)] を選択します。
3. [Configure environment (環境を設定する)] ページの [Provisioning (プロビジョニング)] セクションで [Provision through AWS Proton] を選択します。
4. [Deployment account (デプロイアカウント)] セクションで、この AWS アカウントを選択します。
5. [Environment Settings (環境の設定)] で環境名として「**`my-fargate-environment`**」を入力します。
6. 環境ロールセクションで、新しいサービスロールを選択するか、 AWS Proton サービスロールを既に作成している場合は、既存のサービスロールを選択します。

    a. [New service role (新しいサービスロール)] を選択して新しいロールを作成します。

        i. [Environment role name (環境ロール名)] として **`MyProtonServiceRole`** を入力します。

        ii. アカウントの管理者権限を持つ AWS Proton サービスロールを作成することに同意するには、チェックボックスをオンにします。

    b. [Existing service role (既存のサービスロール)] を選択して既存のロールを使用します。

        - [Environment role name (環境ロール名)] ドロップダウンフィールドでロールを選択します。

7. [Next (次へ)] を選択します。
8. [Configure custom settings (カスタム設定の構成)] ページで、デフォルト値を使用します。
9. [Next (次へ)] を選択して入力を見直します。
10. [作成] を選択します。

    環境の詳細とステータス、および環境の AWS マネージドタグとカスタマーマネージドタグを表示します。
11. ナビゲーションペインで [Environments (環境)] を選択します。

    新しいページには、ステータスやその他の環境の詳細とともに環境のリストが表示されます。

ステップ 6: オプション - サービスを作成してアプリケーションをデプロイする

1. AWS Proton コンソール を開きます。
2. ナビゲーションペインで [Services (サービス)] を選択します。
3. [Services (サービス)] ページで [Create service (サービスの作成)] を選択します。
4. [Choose a service template (サービステンプレートを選択する)] ページでテンプレートカードの右上にあるラジオボタンを選択して My Fargate Service テンプレートを選択します。
5. ページの右下にある [Configure (設定)] を選択します。
6. [Configure service (サービスを構成する)] ページで [Service settings (サービス設定)] セクションにサービス名 **`my-service`** を入力します。
7. (オプション) サービスの説明を入力します。
8. [Service repository settings (サービスリポジトリ設定)] セクションで、次のように操作します。
    a. [CodeStar connection (CodeStar 接続)] でリストから接続を選択します。
    b. [Repository name (リポジトリ名)] でソースコードが含まれているリポジトリの名前を選択します。
    c. [Branch name (ブランチ名)] でソースコードが含まれているリポジトリの名前を選択します。
9. (オプション) [Tags (タグ)] セクションで [Add new tag (新しいタグを追加)] を選択し、キーと値を入力してカスタマーマネージドタグを作成します。次いで、[Next (次へ)] を選択します。

10. [] (カスタム設定を構成する) ページで次の手順に従って、[Service instance (サービスインスタンス)] セクションの [New instance (新しいインスタンス)] セクションでサービスインスタンスパラメータのカスタム値を指定します。

    a. インスタンス名 **my-app-service** を入力します。

    b. サービスインスタンス用に環境 **my-fargate-environment** を選択します。

    c. 残りのインスタンスパラメータをデフォルトのままにします。

    d. [Pipeline inputs (パイプライン入力)] をデフォルトのままにします。

    e. [Next (次へ)] を選択して入力を見直します。

    f. [Create (作成)] を選択してサービスのステータスと詳細を表示します。

11. サービスの詳細ページで、[Overview (概要)] タブと [Pipeline (パイプライン)] タブを選択してサービスインスタンスとパイプラインのステータスを確認してください。これらのページでは、AWS とカスタマーマネージドタグを表示することもできます。 は AWS マネージドタグ AWS Proton を自動的に作成します。[Manage tags (タグの管理)] を選択して、カスタマーマネージドタグを作成や変更します。タグ付けの詳細については、「AWS Proton リソースとタグ付け」を参照してください。

12. サービスが [Active (アクティブ)] になったら、[Overview (概要)] タブの [Service Instances (サービスインスタンス)] セクションで、自分のサービスインスタンスの名前 my-app-service を選択します。

    サービスインスタンスの詳細ページが表示されます。

13. アプリケーションを表示するには、[Output (出力)] セクションで ServiceEndpoint リンクをブラウザにコピーします。

    ウェブページに AWS Proton グラフィックが表示されます。

14. サービスを作成したら、ナビゲーションペインで、[Services (サービス)] を選択してサービスのリストを表示します。

ステップ 7: クリーンアップする。

1. AWS Proton コンソール を開きます。
2. サービスを削除する (サービスを作成した場合)

    a. ナビゲーションペインで [Services (サービス)] を選択します。

    b. [Services (サービス)] ページで、サービス名 my-service を選択します。

画面は my-service のサービス詳細ページに移行します。

c. ページの右上にある [Actions (アクション)] メニューで [Delete (削除)] を選択します。

d. モーダルから Delete アクションの確認を求めるプロンプトが表示されます。

e. 指示に従って操作し、[Yes, delete (はい、削除します)] を選択します。

3. 環境を削除する

a. ナビゲーションペインで [Environments (環境)] を選択します。

b. [Environments (環境)] ページで、先ほど作成した環境の左側にあるラジオボタンを選択します。

c. [Actions (アクション)] を選択してから [Delete (削除)] を選択します。

d. モーダルから Delete アクションの確認を求めるプロンプトが表示されます。

e. 指示に従って操作し、[Yes, delete (はい、削除します)] を選択します。

4. サービステンプレートを削除する

a. ナビゲーションペインで [Service templates (サービステンプレート)] を選択します。

b. [Service templates (サービステンプレート)] ページで、サービステンプレート my-svc-template の左側にあるラジオボタンを選択します。

c. [Actions (アクション)] を選択してから [Delete (削除)] を選択します。

d. モーダルから Delete アクションの確認を求めるプロンプトが表示されます。

e. 指示に従って操作し、[Yes, delete (はい、削除します)] を選択します。これでサービステンプレートとそのすべてのバージョンが削除されます。

5. 環境テンプレートを削除する

a. ナビゲーションペインで [Environment templates (環境テンプレート)] を選択します。

b. [Environment templates (環境テンプレート)] ページで、環境テンプレート my-env-template の左側にあるラジオボタンを選択します。

c. [Actions (アクション)] を選択してから [Delete (削除)] を選択します。

d. モーダルから Delete アクションの確認を求めるプロンプトが表示されます。

e. 指示に従って操作し、[Yes, delete (はい、削除します)] を選択します。これで環境テンプレートとそのすべてのバージョンが削除されます。

6. あなたの Codestar 接続を削除します。

の開始方法 AWS CLI

AWS Proton の使用を開始するには AWS CLI、このチュートリアルに従います。このチュートリアルでは、 に基づくパブリック向け負荷分散 AWS Proton サービスを示します AWS Fargate。このチュートリアルでは、画像が表示された静的 Web サイトをデプロイする CI/CD パイプラインもプロビジョニングします。

始める前に、正しく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、「the section called “前提条件”」を参照してください。

ステップ 1: 環境テンプレートを作成する

このステップでは、管理者として、Amazon Elastic Container Service (Amazon ECS) クラスターと、2 つのパブリック/プライベートサブネットを持った Amazon Virtual Private Cloud (Amazon VPC) があるサンプル環境テンプレートを登録します。

環境テンプレートを登録する

1. AWS Proton サンプル CloudFormation テンプレートリポジトリをあなたの GitHub アカウントまたは組織にフォークします。このリポジトリには、このチュートリアルで使用する環境テンプレートとサービステンプレートが含まれます。

   次に、フォークされたリポジトリを に登録します AWS Proton。詳細については、「the section called “レポジトリを作成する”」を参照してください。
2. 環境テンプレートを作成します。

   環境テンプレートリソースは環境テンプレートのバージョンを追跡します。

   ```
   $ aws proton create-environment-template \
     --name "fargate-env" \
     --display-name "Public VPC Fargate" \
     --description "VPC with public access and ECS cluster"
   ```
3. テンプレート同期設定を作成する。

   AWS Proton は、リポジトリと環境テンプレート間の同期関係を設定します。次に、DRAFT ステータスのテンプレートバージョン 1.0 を作成します。

```
$ aws proton create-template-sync-config \
  --template-name "fargate-env" \
  --template-type "ENVIRONMENT" \
  --repository-name "your-forked-repo" \
  --repository-provider "GITHUB" \
  --branch "your-branch" \
  --subdirectory "environment-templates/fargate-env"
```

4. 環境テンプレートのバージョンが正常に登録されるまでお待ちください。

   このコマンドが `0` の終了ステータスで戻ると、バージョン登録は完了です。これはスクリプトで次のステップでコマンドを正常に実行できるようにするのに便利です。

```
$ aws proton wait environment-template-version-registered \
  --template-name "fargate-env" \
  --major-version "1" \
  --minor-version "0"
```

5. 環境テンプレートをパブリッシュして環境作成で使用可能にします。

```
$ aws proton update-environment-template-version \
  --template-name "fargate-env" \
  --major-version "1" \
  --minor-version "0" \
  --status "PUBLISHED"
```

ステップ 2: サービステンプレートを登録する

このステップでは、あなたは管理者として、ロードバランサーと AWS CodePipelineを使用する CI/CD パイプラインの背後に Amazon ECS Fargate サービスをプロビジョニングするために必要なすべてのリソースを含むサンプルサービステンプレートを登録します。

サービステンプレートを登録する

1. サービステンプレートを作成します。

   サービステンプレートリソースは、サービステンプレートのバージョンを追跡します。

```
$ aws proton create-service-template \
  --name "load-balanced-fargate-svc" \
```

```
  --display-name "Load balanced Fargate service" \
  --description "Fargate service with an application load balancer"
```

2. テンプレート同期設定を作成する。

   AWS Proton は、リポジトリとサービステンプレート間の同期関係を設定します。次に、DRAFT ステータスのテンプレートバージョン 1.0 を作成します。

```
$ aws proton create-template-sync-config \
  --template-name "load-balanced-fargate-svc" \
  --template-type "SERVICE" \
  --repository-name "your-forked-repo" \
  --repository-provider "GITHUB" \
  --branch "your-branch" \
  --subdirectory "service-templates/load-balanced-fargate-svc"
```

3. サービステンプレートバージョンが正常に登録されるまでお待ちください。

   このコマンドが `0` の終了ステータスで戻ると、バージョン登録は完了です。これはスクリプトで次のステップでコマンドを正常に実行できるようにするのに便利です。

```
$ aws proton wait service-template-version-registered \
  --template-name "load-balanced-fargate-svc" \
  --major-version "1" \
  --minor-version "0"
```

4. サービステンプレートをパブリッシュしてサービスの作成で使用可能にします。

```
$ aws proton update-service-template-version \
  --template-name "load-balanced-fargate-svc" \
  --major-version "1" \
  --minor-version "0" \
  --status "PUBLISHED"
```

ステップ 3: 環境をデプロイする

このステップでは、管理者は AWS Proton 環境テンプレートから環境をインスタンス化します。

環境をデプロイするには

1. 登録した環境テンプレートのサンプル仕様ファイルを入手してください。

このファイル environment-templates/fargate-env/spec/spec.yaml は、テンプレートサンプルリポジトリからダウンロードできます。または、リポジトリ全体をローカルで取得し、その environment-templates/fargate-env ディレクトリから create-environment コマンドを実行することもできます。

2. 環境を作成します。

   AWS Proton は、環境仕様から入力値を読み取り、環境テンプレートと組み合わせ、 AWS Proton サービスロールを使用して AWS アカウント内の環境リソースをプロビジョニングします。

```
$ aws proton create-environment \
  --name "fargate-env-prod" \
  --template-name "fargate-env" \
  --template-major-version 1 \
  --proton-service-role-arn "arn:aws:iam::123456789012:role/AWS ProtonServiceRole"
 \
  --spec "file://spec/spec.yaml"
```

3. 環境が正常にデプロイされるまでお待ちください。

```
$ aws proton wait environment-deployed --name "fargate-env-prod"
```

ステップ 4: サービスをデプロイする [アプリケーション開発者]

前のステップでは、管理者がサービステンプレートを登録してパブリッシュし、環境をデプロイしました。アプリケーション開発者として、 AWS Proton サービスを作成して AWS Proton 環境にデプロイできるようになりました。

サービスをデプロイする

1. 管理者が登録したサービステンプレートのサンプル仕様ファイルを入手してください。

   このファイル service-templates/load-balanced-fargate-svc/spec/spec.yaml はテンプレートサンプルリポジトリからダウンロードできます。または、リポジトリ全体をローカルで取得し、その service-templates/load-balanced-fargate-svc ディレクトリから create-service コマンドを実行することもできます。

2. AWS Proton サンプルサービスリポジトリをあなたの GitHub アカウントまたは組織にフォークします。このリポジトリには、このチュートリアルで使用するアプリケーションソースコードが含まれています。
3. サービスを作成します。

   AWS Proton は、サービス仕様から入力値を読み取り、サービステンプレートと組み合わせ、仕様で指定された環境のアカウント AWS でサービスリソースをプロビジョニングします。 AWS CodePipeline パイプラインは、 コマンドで指定したリポジトリからアプリケーションコードをデプロイします。

   ```
   $ aws proton create-service \
     --name "static-website" \
     --repository-connection-arn \
       "arn:aws:codestar-connections:us-east-1:123456789012:connection/your-codestar-connection-id" \
     --repository-id "your-GitHub-account/aws-proton-sample-services" \
     --branch-name "main" \
     --template-major-version 1 \
     --template-name "load-balanced-fargate-svc" \
     --spec "file://spec/spec.yaml"
   ```
4. サービスが停止するのを待ちます。

   ```
   $ aws proton wait service-created --name "static-website"
   ```
5. 出力を取得し、あなたの新しい Web サイトを表示します。

   次のコマンドを実行します。

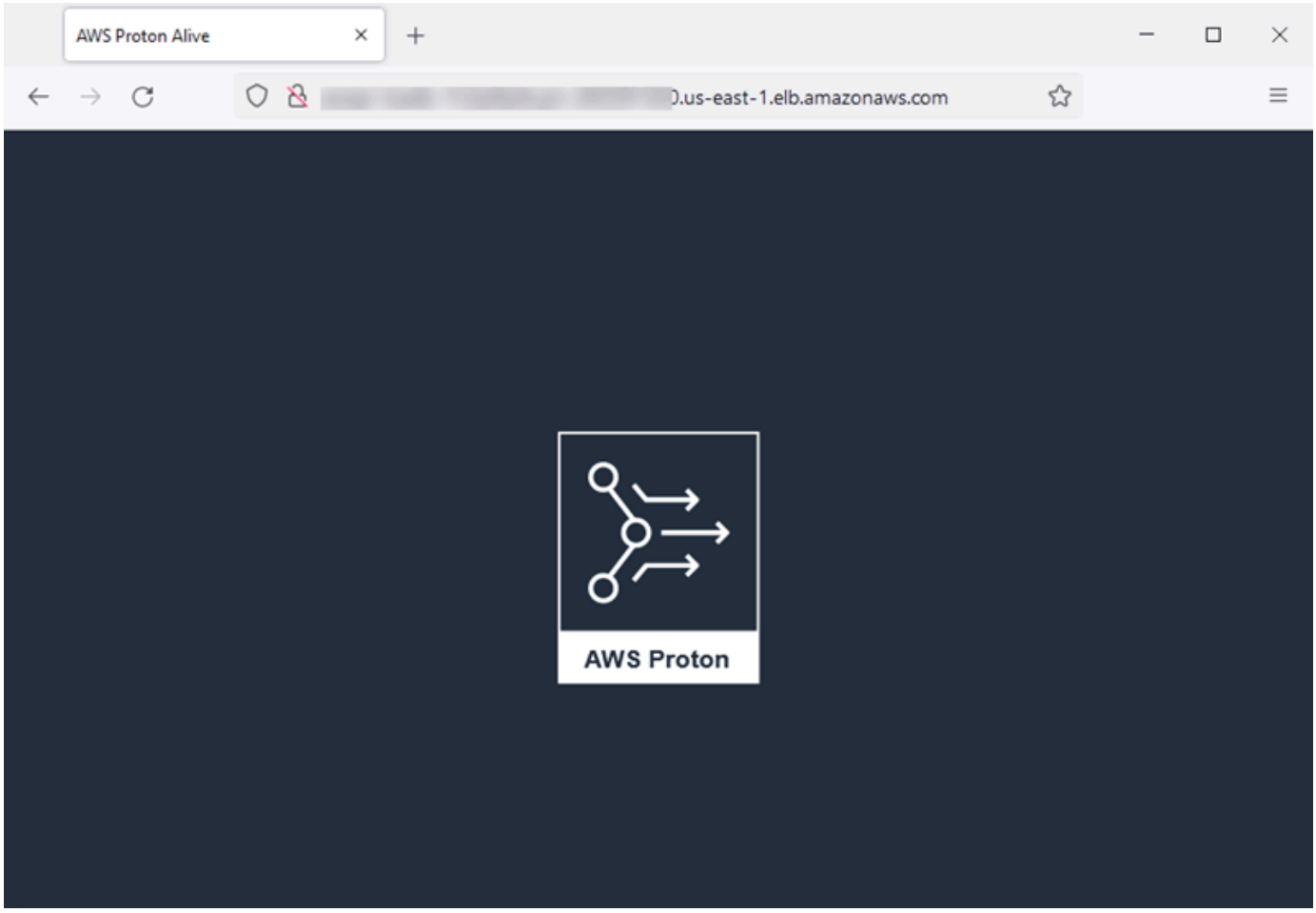
   ```
   $ aws proton list-service-instance-outputs \
     --service-name "static-website" \
     --service-instance-name load-balanced-fargate-svc-prod
   ```

   コマンドの出力は、次のようになります。

   ```
   {
       "outputs": [
           {
               "key": "ServiceURL",
               "valueString": "http://your-service-endpoint.us-east-1.elb.amazonaws.com"
   ```

```
        }
    ]
}
```

ServiceURL インスタンス出力の値は、あなたの新しいサービス Web サイトのエンドポイントです。あなたのブラウザを使用してそのサイトに移動します。では、次のタイプのパラメータを使用できます。



ステップ 5: クリーンアップする (オプション)

このステップでは、このチュートリアルの一部として作成した AWS リソースを調べ、これらのリソースに関連するコストを節約するために、リソースを削除します。

チュートリアルのリソースを削除する

1. サービスを消去するには、次のコマンドを実行します。

```
$ aws proton delete-service --name "static-website"
```

2. 環境を削除するには、次のコマンドを実行します 。

```
$ aws proton delete-environment --name "fargate-env-prod"
```

3. テンプレートを削除するには、次のコマンドを実行します。

```
$ aws proton delete-template-sync-config \
  --template-name "load-balanced-fargate-svc" \
  --template-type "SERVICE"
$ aws proton delete-service-template --name "load-balanced-fargate-svc"
```

4. 環境テンプレートを削除するには、次のコマンドを実行します。

```
$ aws proton delete-template-sync-config \
  --template-name "fargate-env" \
  --template-type "ENVIRONMENT"
$ aws proton delete-environment-template --name "fargate-env"
```

AWS Proton テンプレートライブラリ

AWS Proton チームは、GitHub でテンプレート例のライブラリを管理しています。このライブラリには、多くの一般的な環境およびアプリケーション・インフラストラクチャ・シナリオのためのコードとしてのinfrastructure as code (IaC)ファイルの例が含まれています。

テンプレートライブラリは次の 2 つの GitHub リポジトリに保存されています。

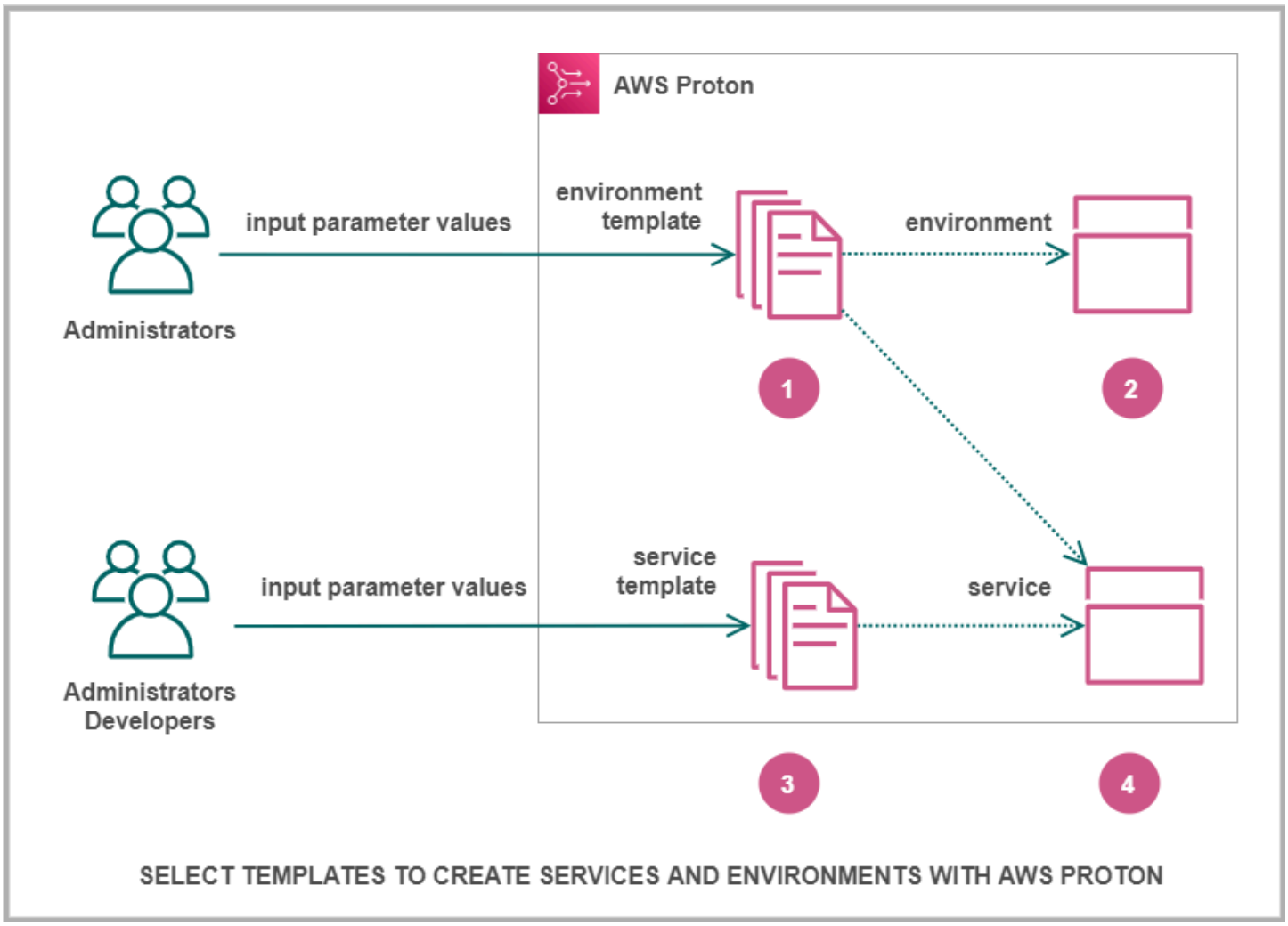
- aws-proton-cloudformation-sample-templates — AWS CloudFormation で Jinja を IaC 言語として使用するテンプレートバンドルの例。これらの例はAWSマネージドプロビジョニング環境に使用できます。
- aws-proton-terraform-sample-templates — IaC 言語として Terraform を使用するテンプレートバンドルの例。これらの例は セルフマネージド型のプロビジョニング 環境に使用できます。

これらのリポジトリにはそれぞれ、リポジトリの内容と構造に関するすべての情報が記載された README ファイルがあります。それぞれの例には、テンプレートがカバーするユースケース、例のアーキテクチャ、テンプレートが取る入力パラメータに関する情報が含まれています。

ライブラリのリポジトリの 1 つを、あなたの GitHub アカウントにフォークすることで、このライブラリのテンプレートを直接使用できます。あるいは、これらの例をあなたの環境とサービステンプレートを開発するための出発点として使用することもできます。

の AWS Proton 仕組み

では AWS Proton、環境をプロビジョニングしてから、それらの環境で実行されているサービスをプロビジョニングします。環境とサービスは、 AWS Proton それぞれバージョン管理されたテンプレートライブラリで選択した環境テンプレートとサービステンプレートに基づいています。



1

管理者として で環境テンプレートを選択するときは AWS Proton、必要な入力パラメータの値を指定します。

2

AWS Proton は環境テンプレートとパラメータ値を使用して環境をプロビジョニングします。

3

開発者または管理者として、 でサービステンプレートを選択するときは AWS Proton、必要な入力パラメータの値を指定します。アプリケーションまたはサービスをデプロイする環境も選択します。

4

AWS Proton は、サービステンプレート、サービスと選択した環境パラメータ値の両方を使用してサービスをプロビジョニングします。

入力パラメータの値を指定することで再利用するテンプレートおよび複数のユースケース、アプリケーション、またはサービスをカスタマイズできます。

これを機能させるには、環境またはサービステンプレートバンドルを作成し、登録済みの環境またはサービステンプレートにそれぞれアップロードします。

テンプレートバンドルには、環境またはサービスをプロビジョニング AWS Proton するために必要なものがすべて含まれています。

環境やサービスのテンプレートを作成するとき、 AWS Proton が環境やサービスのプロビジョニングに使用するパラメトリック Infrastructure as Code (IaC) ファイルがあるテンプレートバンドルをアップロードします。

環境またはサービステンプレートを選択して環境またはサービスを作成または更新する場合、テンプレートバンドル iaC ファイルパラメータの値を指定します。

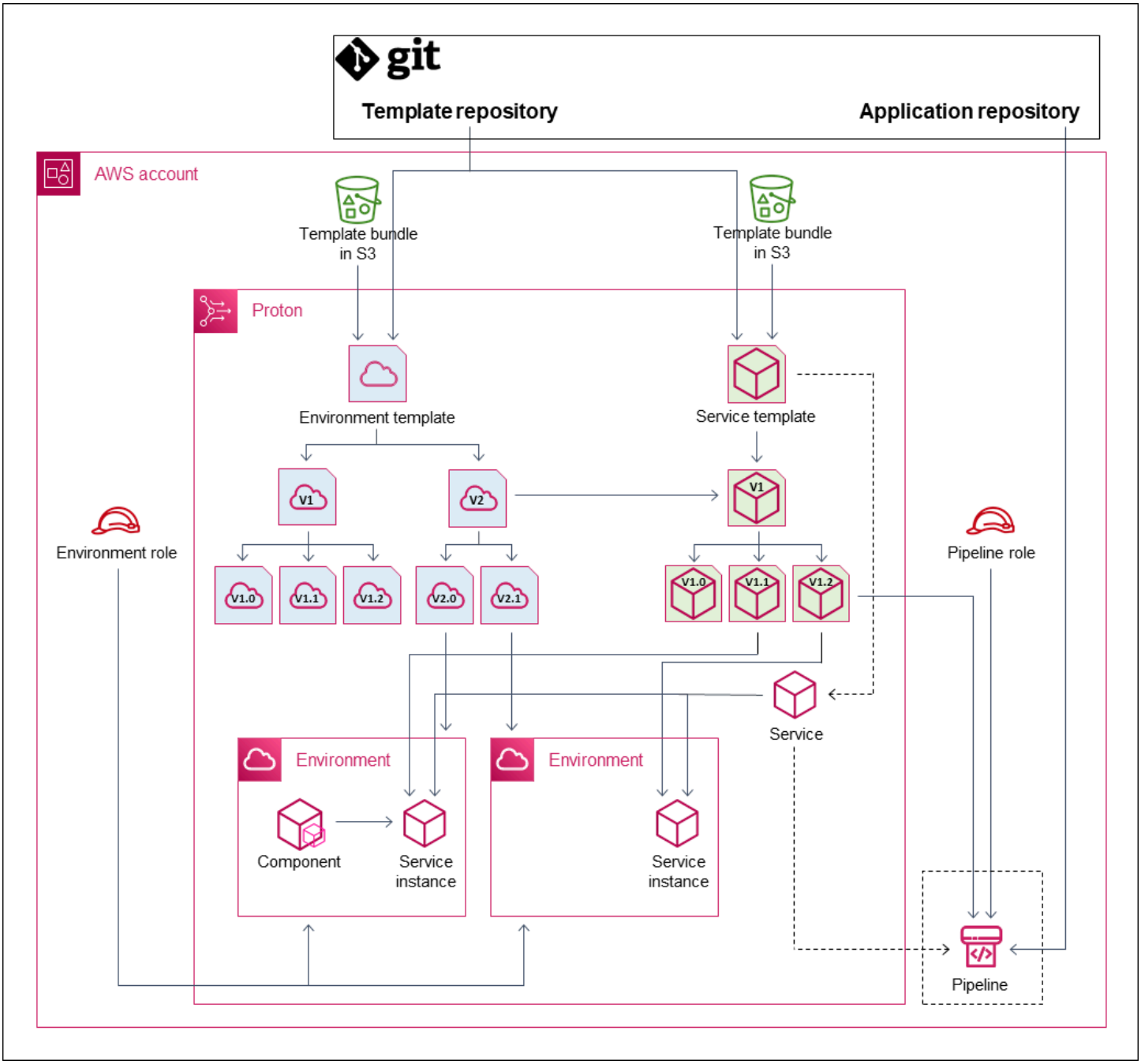
トピック

- AWS Proton オブジェクト
- がインフラストラクチャを AWS Proton プロビジョニングする方法
- AWS Proton 用語

AWS Proton オブジェクト

次の図は、メイン AWS Proton オブジェクトとその他のオブジェクト AWS やサードパーティーオブジェクトとの関係を示しています。矢印はデータフローの方向 (依存関係の逆方向) を表しています。

これらの AWS Proton オブジェクトの簡単な説明とリファレンスリンクを含む図に従います。



- 環境テンプレート — AWS Proton 環境の作成に使用できる環境テンプレートバージョンのコレクションです。

  詳細については、「テンプレートのオーサリングとバンドル」および「テンプレート」を参照してください。
- 環境テンプレートバージョン — 環境テンプレートの特定のバージョンです。S3 バケットまたは Git リポジトリからテンプレートバンドルを入力として受け取ります。バンドルは、 AWS Proton 環境の Infrastructure as Code (IaC) および関連する入力パラメータを指定します。

詳細については the section called “バージョン”、the section called “配信”、および the section called “テンプレートの同期設定”を参照してください。

- 環境 – AWS Proton サービスがデプロイされる一連の共有 AWS インフラストラクチャリソースとアクセスポリシー。 AWS リソースは、特定のパラメータ値で呼び出される環境テンプレートバージョンを使用してプロビジョニングされます。アクセスポリシーはサービスロールで提供されます。

  詳細については、「環境」を参照してください。

- サービステンプレート — AWS Proton サービスの作成に使用できるサービステンプレートバージョンのコレクションです。

  詳細については、「テンプレートのオーサリングとバンドル」および「 テンプレート」を参照してください。

- サービステンプレートのバージョン - サービステンプレートの特定のバージョン。S3 バケットまたは Git リポジトリからテンプレートバンドルを入力として受け取ります。バンドルは、 AWS Proton サービスの Infrastructure as Code (IaC) および関連する入力パラメータを指定します。

  サービステンプレートバージョンでは、バージョンに基づいてサービスインスタンスに対する以下の制約も指定します。

  - 互換性のある環境テンプレート — インスタンスは、互換性のある環境テンプレートに基づく環境のみで実行できます。
  - サポートされているコンポーネントソース — 開発者がインスタンスに関連付けることができるコンポーネントのタイプ。

  詳細については、「the section called “バージョン”」、「the section called “配信”」、および「the section called “テンプレートの同期設定”」を参照してください。

- サービス — サービステンプレートで指定したリソースを使用してアプリケーションを実行するサービスインスタンスのコレクションであり、オプションとして、アプリケーションコードをこれらのインスタンスにデプロイする CI/CD パイプラインがあります。

  この図では、サービステンプレートからの破線は、サービスがサービスインスタンスとパイプラインにテンプレートを渡していることを示しています。

  詳細については、「サービス」を参照してください。

- サービスインスタンス – 特定の AWS Proton 環境でアプリケーションを実行する AWS インフラストラクチャリソースのセット。 AWS リソースは、特定のパラメータ値で呼び出されるサービステンプレートバージョンを使用してプロビジョニングされます。

  詳細については、「サービス」および「the section called “インスタンスを更新する”」を参照してください。
- パイプライン — サービスのインスタンスにアプリケーションをデプロイするオプションの CI/CD パイプライン。このパイプラインをプロビジョニングするためのアクセスポリシーも付いています。アクセスポリシーはサービスロールで提供されます。サービスには常にパイプラインが関連付け AWS Proton られているわけではありません。 の外部でアプリケーションコードのデプロイを管理できます AWS Proton。

  この図では、Service の破線と Pipeline の周りの破線は、CI/CD デプロイを自分で管理することを選択した場合、 AWS Proton パイプラインが作成されず、独自のパイプラインが AWS アカウント内にない可能性があることを意味します。

  詳細については、「サービス」および「the section called “パイプラインを更新する”」を参照してください。
- コンポーネント — 開発者が定義したサービスインスタンスの拡張です。環境とサービスインスタンスによって提供されるリソースに加えて、特定のアプリケーションに必要な追加の AWS インフラストラクチャリソースを指定します。プラットフォームチームは、コンポーネントがプロビジョニングできるインフラストラクチャを、コンポーネントロールを環境にアタッチして制御します。

  詳細については、「コンポーネント」を参照してください。

がインフラストラクチャを AWS Proton プロビジョニングする方法

AWS Proton は、次のいずれかの方法でインフラストラクチャをプロビジョニングできます。

- AWSマネージドプロビジョニング – ユーザーに代わってプロビジョニングエンジンを AWS Proton 呼び出します。この方法は AWS CloudFormation テンプレートバンドルのみをサポートします。詳細については、「the section called “CloudFormation IaC ファイル”」を参照してください。
- CodeBuild プロビジョニング – 指定したシェルコマンドを実行する AWS CodeBuild ために AWS Proton を使用します。コマンドは、 AWS Proton が提供する入力を読み取ることができ、インフラストラクチャのプロビジョニングまたはプロビジョニング解除と出力値の生成を担当します。こ

の方法のテンプレートバンドルには、マニフェストファイル内のあなたのコマンドと、これらのコマンドで必要になるプログラム、スクリプト、またはその他のファイルが含まれます。

CodeBuild プロビジョニングを使用する例として、 を使用して AWS リソース AWS Cloud Development Kit (AWS CDK) をプロビジョニングするコードと、CDK をインストールして CDK コードを実行するマニフェストを含めることができます。

詳細については、「the section called “CodeBuild バンドル”」を参照してください。

> **Note**
>
> CodeBuild プロビジョニングは環境とサービスで使用できます。現時点では、この方法でコンポーネントをプロビジョニングすることはできません。

- セルフマネージドプロビジョニング – 指定したリポジトリにプルリクエスト (PR) AWS Proton を発行し、独自のインフラストラクチャデプロイシステムがプロビジョニングプロセスを実行します。この方法は Terraform テンプレートバンドルのみをサポートします。詳細については、「the section called “Terraform IaC ファイル”」を参照してください。

AWS Proton は、環境とサービスのプロビジョニング方法を個別に決定して設定します。環境またはサービスを作成または更新するとき、 AWS Proton はあなたが提供したテンプレートバンドルを調べ、テンプレートバンドルが示すプロビジョニング方法を決定します。環境レベルでは、AWS Identity and Access Management (IAM) ロール、環境アカウント接続、インフラストラクチャリポジトリなど、環境とその潜在的なサービスがプロビジョニング方法に必要なパラメータを指定します。

AWS Proton を使用してサービスをプロビジョニングするデベロッパーは、プロビジョニング方法に関係なく同じエクスペリエンスが得られます。開発者はプロビジョニング方法を意識する必要はなく、サービスのプロビジョニングプロセスのいかなる要素も変更する必要はありません。サービステンプレートはプロビジョニング方法を設定します。また、開発者がサービスをデプロイする各環境は、サービスインスタンスのプロビジョニングに必要なパラメータを提供します。

次の図は、さまざまなプロビジョニング方法の主な特徴をまとめたものです。表の後のセクションでは、各方法の詳細を説明します。

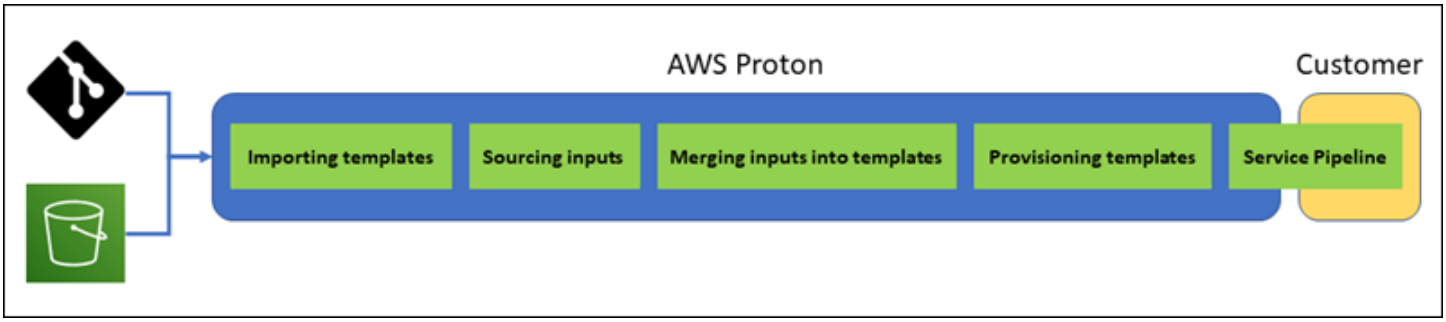
プロビジョニング方法	テンプレート	プロビジョニング実行者	ステータス追跡者
AWSによる管理	マニフェスト、スキーマ、IaC ファイル (CloudFormation)	AWS Proton (CloudFormation 経由)	AWS Proton (CloudFormation 経由)
CodeBuild	マニフェスト (コマンドを使用) 、スキーマ、コマンドの依存関係 (AWS CDK コードなど)	AWS Proton (CodeBuild 経由)	AWS Proton (コマンドは CodeBuild を介してステータスを返します)
自己管理	マニフェスト、スキーマ、IaC ファイル (Terraform)	コード (Git アクション経由)	コード (API コール AWS を介して に渡されます)

AWSマネージドプロビジョニングの仕組み

環境またはサービスが AWSマネージドプロビジョニングを使用する場合、インフラストラクチャは次のようにプロビジョニングされます。

1. AWS Proton 顧客 (管理者または開発者) が AWS Proton リソース (環境またはサービス) を作成します。カスタマーは、リソースのテンプレートを選択し、必要なパラメータを指定します。詳細については、次の「the section called “AWSマネージドプロビジョニングに関する考慮事項”」セクションを参照してください。
2. AWS Proton は、リソースをプロビジョニングするための完全な CloudFormation テンプレートをレンダリングします。
3. AWS Proton を呼び出し CloudFormation て、レンダリングされたテンプレートを使用してプロビジョニングを開始します。
4. AWS Proton は CloudFormation デプロイを継続的にモニタリングします。
5. プロビジョニングが完了すると、 は障害発生時にエラー AWS Proton を報告し、Amazon VPC ID などのプロビジョニング出力を成功時にキャプチャします。

次の図は、 AWS Proton がこれらのステップのほとんどを直接処理することを示しています。



AWSマネージドプロビジョニングに関する考慮事項

- インフラストラクチャプロビジョニングロール – 環境またはそこで実行されているサービスインスタンスが AWSマネージドプロビジョニングを使用する場合、管理者は (直接または AWS Proton 環境アカウント接続の一部として) IAM ロールを設定する必要があります。 はこのロール AWS Proton を使用して、これらの AWSマネージドプロビジョニングリソースのインフラストラクチャをプロビジョニングします。ロールには、これらのリソースのテンプレートに含まれるすべてのリソースを作成 CloudFormation するために使用するアクセス許可が必要です。

  詳細については、「the section called “IAM ロール”」および「the section called “サービスロールポリシーの例”」を参照してください。
- サービスプロビジョニング – 開発者が AWSマネージドプロビジョニングを使用するサービスインスタンスを環境にデプロイすると、 はその環境に提供されるロール AWS Proton を使用してサービスインスタンスのインフラストラクチャをプロビジョニングします。開発者にはこのロールは表示されず、変更することもできません。
- パイプラインを使用したサービス – AWSマネージドプロビジョニングを使用するサービステンプレートには、YAML CloudFormation スキーマに書き込まれたパイプライン定義が含まれる場合があります。 AWS Proton また、 は を呼び出してパイプラインを作成します CloudFormation。がパイプラインの作成 AWS Proton に使用するロールは、個々の環境のロールとは別です。このロールは、 AWS アカウントレベルで 1 回のみ AWS Proton 個別に に提供され、オール AWSマネージドパイプラインのプロビジョニングと管理に使用されます。このロールには、パイプラインやパイプラインに必要なその他のリソースを作成する権限が必要です。

  次の手順では、 AWS Protonにパイプラインロールを指定する方法を示します。

  AWS Proton console

  パイプラインロールを指定する

  1. AWS Proton コンソールのナビゲーションペインで、[設定] > [アカウント設定] を選択し、[設定] を選択します。

2. Pipeline AWSマネージドロールセクションを使用して、 AWSマネージドプロビジョニング用の新規または既存のパイプラインロールを設定します。

AWS Proton API

パイプラインロールを提供する

1. UpdateAccountSettings API アクションを使用してください。
2. `pipelineServiceRoleArn` パラメータでパイプラインサービスロールの Amazon リソースネーム (ARN) を指定します。

AWS CLI

パイプラインロールを指定する

次のコマンドを実行してください。

```
$ aws proton update-account-settings \
    --pipeline-service-role-arn \
        "arn:aws:iam::123456789012:role/my-pipeline-role"
```

CodeBuild プロビジョニングの働き

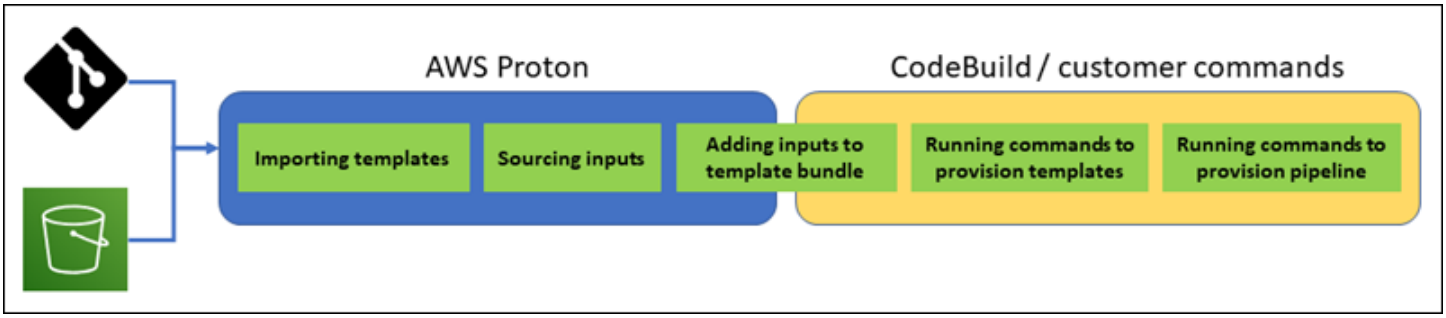
環境またはサービスで CodeBuild プロビジョニングが使用される場合、インフラストラクチャは次のようにプロビジョニングされます。

1. AWS Proton 顧客 (管理者または開発者) が AWS Proton リソース (環境またはサービス) を作成します。カスタマーは、リソースのテンプレートを選択し、必要なパラメータを指定します。詳細については、次の「the section called “CodeBuild プロビジョニングに関する考慮事項”」セクションを参照してください。
2. AWS Proton は、リソースをプロビジョニングするための入力パラメータ値を含む入力ファイルをレンダリングします。
3. AWS Proton は CodeBuild を呼び出してジョブを開始します。CodeBuild ジョブは、テンプレートで指定されたカスタマーシェルコマンドを実行します。これらのコマンドは必要なインフラストラクチャをプロビジョニングし、オプションで入力値を読み取ります。

4. プロビジョニングが完了すると、最後のカスタマーコマンドはプロビジョニングステータスを CodeBuild に返し、NotifyResourceDeploymentStatusChange AWS Proton API アクションを呼び出して、Amazon VPC ID などの出力があればそれを提供します。

> **Important**
>
> コマンドでプロビジョニングステータスが正しく CodeBuild に返り、出力が提供されたことを確認してください。そうでない場合、 AWS Proton はプロビジョニングステータスを適切に追跡できず、サービスインスタンスに正しい出力を提供できません。

次の図は、 が AWS Proton 実行するステップと、CodeBuild ジョブ内でコマンドが実行するステップを示しています。



CodeBuild プロビジョニングに関する考慮事項

- インフラストラクチャプロビジョニングロール – 環境またはそこで実行されているサービスインスタンスが CodeBuild ベースのプロビジョニングを使用する場合、管理者は (直接または AWS Proton 環境アカウント接続の一部として) IAM ロールを設定する必要があります。 はこのロール AWS Proton を使用して、これらの CodeBuild プロビジョニングリソースのインフラストラクチャをプロビジョニングします。ロールには、CodeBuild を使用して、これらのリソースのテンプレート内のコマンドによって提供されるすべてのリソースを作成するための権限が必要です。

  詳細については、「the section called “IAM ロール”」および「the section called “サービスロールポリシーの例”」を参照してください。
- サービスプロビジョニング – 開発者が CodeBuild プロビジョニングを使用するサービスインスタンスを環境にデプロイすると、 はその環境に提供されるロール AWS Proton を使用してサービスインスタンスのインフラストラクチャをプロビジョニングします。開発者にはこのロールは表示されず、変更することもできません。

- パイプラインを使用したサービス – CodeBuild プロビジョニングを使用するサービステンプレートには、パイプラインをプロビジョニングするためのコマンドが含まれている場合があります。AWS Proton また、 は CodeBuild を呼び出してパイプラインを作成します。がパイプラインの作成 AWS Proton に使用するロールは、個々の環境のロールとは別です。このロールは、 AWS アカウントレベルで 1 回のみ AWS Proton 個別に に提供され、すべての CodeBuild ベースのパイプラインのプロビジョニングと管理に使用されます。このロールには、パイプラインやパイプラインに必要なその他のリソースを作成する権限が必要です。

  次の手順では、 AWS Protonにパイプラインロールを指定する方法を示します。

  AWS Proton console

  パイプラインロールを指定する

  1. AWS Proton コンソールのナビゲーションペインで、[設定] > [アカウント設定] を選択し、[設定] を選択します。
  2. Codebuild パイプラインプロビジョニングロールセクションを使用して、CodeBuild プロビジョニング用の新規または既存のパイプラインロールを設定します。

  AWS Proton API

  パイプラインロールを提供するには

  1. UpdateAccountSettings API アクションを使用してください。
  2. `pipelineCodebuildRoleArn` パラメータでパイプラインサービスロールの Amazon リソースネーム (ARN) を指定します。

  AWS CLI

  パイプラインロールを指定する

  次のコマンドを実行してください。

```
$ aws proton update-account-settings \
    --pipeline-codebuild-role-arn \
        "arn:aws:iam::123456789012:role/my-pipeline-role"
```

セルフマネージド型のプロビジョニングの働き

セルフマネージド型プロビジョニングを使用するように環境を設定すると、インフラストラクチャは次のようにプロビジョニングされます。

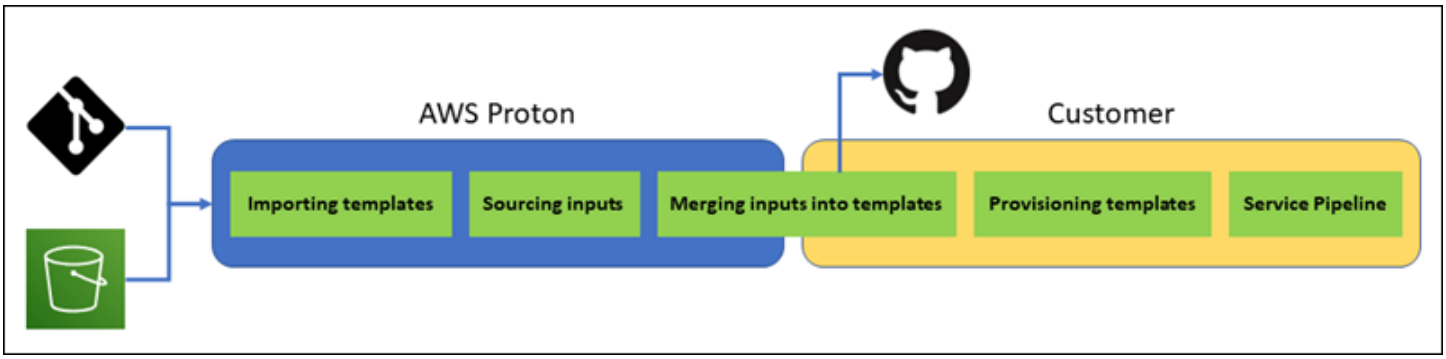
1. AWS Proton 顧客 (管理者または開発者) が AWS Proton リソース (環境またはサービス) を作成します。カスタマーはリソースのテンプレートを選択し、必要なパラメータを指定します。環境については、カスタマーはリンクされたインフラストラクチャリポジトリも提供します。詳細については、次の「the section called “セルフマネージドプロビジョニングに関する考慮事項”」セクションを参照してください。
2. AWS Proton は完全な Terraform テンプレートをレンダリングします。これは、複数のフォルダにある可能性のある 1 つ以上の Terraform ファイルと、この `.tfvars` 変数ファイルへのリソース作成呼び出しで提供される変数 file. AWS Proton writes パラメータ値で構成されます。
3. AWS Proton は、レンダリングされた Terraform テンプレートを使用してインフラストラクチャリポジトリに PR を送信します。
4. カスタマー (管理者または開発者) が PR をマージすると、カスタマーの自動化コードによってプロビジョニングエンジンがトリガーされ、マージされたテンプレートでインフラストラクチャのプロビジョニングが開始されます。

> **Note**
>
> 顧客 (管理者または開発者) が PR を閉じると、 は PR をクローズとして AWS Proton 認識し、デプロイをキャンセル済みとしてマークします。

5. プロビジョニングが完了すると、お客様のオートメーションは NotifyResourceDeploymentStatusChange AWS Proton API アクションを呼び出して、完了を示し、ステータス (成功または失敗) を指定し、Amazon VPC ID などの出力があればそれを提供します。

> **Important**
>
> オートメーションコードがプロビジョニングステータスと出力 AWS Proton で を呼び出すことを確認します。そうでない場合は、プロビジョニングを必要以上に長く保留中と見なし、進行中ステータスを表示し続ける AWS Proton ことができます。

次の図は、 が AWS Proton 実行するステップと、独自のプロビジョニングシステムが実行するステップを示しています。



セルフマネージドプロビジョニングに関する考慮事項

- インフラストラクチャリポジトリ – 管理者がセルフマネージドプロビジョニング用に環境を設定する場合、リンクされたインフラストラクチャリポジトリを提供する必要があります。 はこのリポジトリに PRs AWS Proton を送信して、環境のインフラストラクチャとその環境にデプロイされているすべてのサービスインスタンスをプロビジョニングします。リポジトリ内の顧客所有のオートメーションアクションは、環境およびサービステンプレートに含まれるすべてのリソースを作成するアクセス許可を持つ IAM ロールと、送信先 AWS アカウントを反映する ID を引き受ける必要があります。ロールを引き受ける GitHub Action の例については、GitHub Actions の「認証情報の設定」アクションドキュメントの「ロールの引き受け」を参照してください。 AWS GitHub
- アクセス許可 – プロビジョニングコードは、必要に応じてアカウントで認証し (たとえば、 AWS アカウントへの認証) 、リソースプロビジョニング認可を提供する (たとえば、ロールを提供する) 必要があります。
- サービスプロビジョニング – 開発者がセルフマネージドプロビジョニングを使用するサービスインスタンスを環境にデプロイすると、 は環境に関連付けられたリポジトリに PR AWS Proton を送信して、サービスインスタンスのインフラストラクチャをプロビジョニングします。開発者にはこのリポジトリは表示されず、変更することもできません。

> **Note**
>
> サービスを作成する開発者は、プロビジョニング方法に関係なく同じプロセスを使用するため、違いは取り除かれています。ただし、セルフマネージドプロビジョニングでは、誰か (自分ではない場合もある) がインフラストラクチャリポジトリ内の PR をマージするまでプロビジョニングの開始を待たなければならないため、開発者への応答が遅くなるおそれがあります。

- パイプラインを使用したサービス – セルフマネージドプロビジョニングを使用する環境のサービステンプレートには、Terraform HCL で記述された AWS CodePipeline パイプライン定義 (パイプラインなど) が含まれる場合があります。がこれらのパイプライン AWS Proton をプロビジョニングできるように、管理者はリンクされたパイプラインリポジトリを提供します AWS Proton。パイプラインをプロビジョニングする場合、リポジトリ内の顧客所有のオートメーションアクションは、パイプラインをプロビジョニングするアクセス許可を持つ IAM ロールと、送信先 AWS アカウントを反映する ID を引き受ける必要があります。パイプラインのリポジトリとロールは、個々の環境で使用されるロールからは切り離されています。リンクされたリポジトリは、 AWS アカウントレベルで 1 回のみ AWS Proton 個別に に提供され、すべてのパイプラインのプロビジョニングと管理に使用されます。ロールには、パイプラインやパイプラインに必要なその他のリソースを作成する権限が必要です。

次の手順では、 AWS Protonにパイプラインリポジトリとロールを提供する方法を示します。

AWS Proton console

パイプラインロールを指定する

1. AWS Proton コンソールのナビゲーションペインで、[設定] > [アカウント設定] を選択し、[設定] を選択します。
2. CI/CD パイプラインリポジトリセクションを使用して、新規または既存のリポジトリリンクを設定します。

AWS Proton API

パイプラインロールを提供する

1. UpdateAccountSettings API アクションを使用してください。
2. `pipelineProvisioningRepository` パラメータには、パイプラインリポジトリのプロバイダー、名前、ブランチを指定します。

AWS CLI

パイプラインロールを指定する

次のコマンドを実行してください。

```
$ aws proton update-account-settings \
    --pipeline-provisioning-repository \
```

```
"provider=GITHUB,name=my-pipeline-repo-name,branch=my-branch"
```

- セルフマネージドプロビジョニング済みリソースの削除 — Terraform モジュールには、リソース定義に加えて、Terraform の運用に必要な設定要素が含まれる場合があります。したがって、AWS Proton は環境またはサービスインスタンスのすべての Terraform ファイルを削除することはできません。代わりに、 はファイルを削除対象として AWS Proton マークし、PR メタデータのフラグを更新しました。あなたの自動化機能はそのフラグを読み取り、そのフラグで terraform destroy コマンドをトリガーできます。

AWS Proton 用語

環境テンプレート

複数のアプリケーションまたはリソースで使用される共有インフラストラクチャ (VPC またはクラスターなど) を定義します。

環境テンプレートバンドル

AWS Protonで環境テンプレートを作成し、登録するためにアップロードするファイルのコレクション。テンプレートバンドルには以下が含まれます。

1. Infrastructure as Code 入力パラメータを定義するスキーマファイル。
2. VPC やクラスターなど、複数のアプリケーションまたはリソースで使用される共有インフラストラクチャを定義するコードとしての Infrastructure as Code (IaC) ファイル。
3. IaC ファイルのリストを示すマニフェストファイル。

環境

VPC またはクラスターなど、複数のアプリケーションまたはリソースで使用されるプロビジョニングされた共有インフラストラクチャ。

サービステンプレート

環境内のアプリケーションまたはマイクロサービスをデプロイおよび維持するために必要なインフラストラクチャのタイプを定義します。

サービステンプレートバンドル

AWS Protonでサービステンプレートを作成し、登録するためにアップロードするファイルのコレクション。サービステンプレートバンドルには以下が含まれます。

1. Infrastructure as Code (IaC) 入力パラメータを定義するスキーマファイル。

2. 環境内のアプリケーションまたはマイクロサービスをデプロイおよび維持するうえで必要なインフラストラクチャを定義する IaC ファイル。
3. IaC ファイルのリストを示すマニフェストファイル。
4. オプションです。
   a. サービスパイプラインインフラストラクチャを定義する IaC ファイル。
   b. IaC ファイルのリストを示すマニフェストファイル。

サービス

環境内のアプリケーションやマイクロサービスをデプロイし、維持するうえで必要なプロビジョニングされたインフラストラクチャ。

サービスインスタンス

環境内のアプリケーションまたはマイクロサービスをサポートするプロビジョニングされたインフラストラクチャ。

サービスパイプライン

パイプラインをサポートするプロビジョニングされたインフラストラクチャ。

テンプレートのバージョン

テンプレートのメジャーバージョンまたはマイナーバージョン。詳細については、「バージョン付きテンプレート」を参照してください。

入力パラメータ

スキーマファイルで定義され、Infrastructure as Code (IaC) ファイルとして使用されるので、iaC ファイルはさまざまなユースケースに繰り返し使用できます。

スキーマファイル

Infrastructure as Code ファイル入力パラメータを定義します。

仕様ファイル

スキーマファイルに定義されているように、インフラストラクチャの値をコードファイル入力パラメータとして指定します。

マニフェストファイル

Infrastructure as Code ファイルの一覧を表示します。

のテンプレートの作成とバンドルの作成 AWS Proton

AWS Proton は、Infrastructure as Code (IaC) ファイルに基づいてリソースをプロビジョニングします。インフラストラクチャは再利用可能な IaC ファイルに記述します。ファイルをさまざまな環境やアプリケーションで再利用可能できるように、IaC ファイルはテンプレートとしてオーサリングし、入力パラメータを定義し、それらのパラメータを IaC 定義で使用します。後でプロビジョニングリソース (環境、サービスインスタンス、またはコンポーネント) を作成すると、 はレンダリングエンジン AWS Proton を使用します。レンダリングエンジンは、入力値をテンプレートと組み合わせて、プロビジョニングの準備ができた IaC ファイルを作成します。

管理者はほとんどのテンプレートをテンプレートバンドルとして作成し、アップロードして登録します AWS Proton。このページの残りの部分では、これらの AWS Proton テンプレートバンドルについて説明します。直接定義されたコンポーネントはこの説明の対象外です。それらのコンポーネントは、開発者が作成して IaC テンプレートファイルを直接提供します。コンポーネントの詳細については、「コンポーネント」を参照してください。

トピック

テンプレートバンドル

管理者は、 テンプレートを作成して登録します AWS Proton。これらのテンプレートを使用して環境とサービスを作成します。サービスを作成すると、 はサービスインスタンスを AWS Proton プロビジョニングし、選択した環境にデプロイします。詳細については、「AWS Proton プラットフォームチーム向け」を参照してください。

でテンプレートを作成して登録するには AWS Proton、 と環境またはサービスをプロビジョニング AWS Proton する必要がある Infrastructure as Code (IaC) ファイルを含むテンプレートバンドルをアップロードします。

テンプレートバンドルには以下が含まれます。

- Infrastructure as Code (IaC) ファイルと IaC ファイルのリストを示すマニフェスト YAML ファイル。
- IaC ファイル入力パラメータ定義のためのスキーマ YAML ファイル。

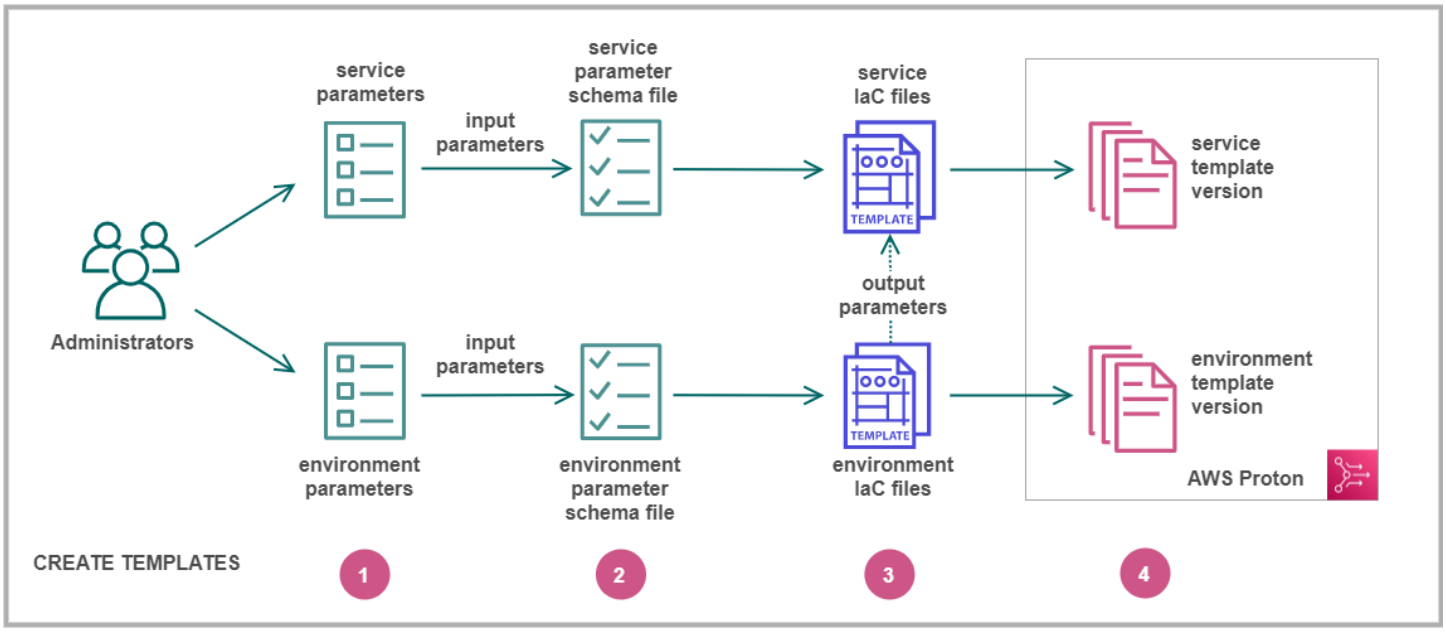
CloudFormation 環境テンプレートバンドルには、1 つの IaC ファイルが含まれます。

CloudFormation サービステンプレートバンドルには、サービスインスタンス定義用の IaC ファイルとパイプライン定義用のオプション IaC ファイルが含まれています。

Terraform 環境およびサービステンプレートバンドルには、それぞれ複数の IaC ファイルを含めることができます。

AWS Proton には入力パラメータスキーマファイルが必要です。 AWS CloudFormation を使用して IaC ファイルを作成するときは、Jinja 構文を使用して入力パラメータを参照します。 は、IaC ファイル内のパラメータを参照するために使用できるパラメータ名前空間 AWS Proton を提供します。

次の図は、テンプレートを作成するために実行できるステップの例を示しています AWS Proton。



1

は入力パラメータを識別します。

2

あなたの入力パラメータを定義するスキーマファイルを作成します。

3

あなたの入力パラメータを参照する IaC ファイルを作成します。環境 IaC ファイルの出力をサービス IaC ファイルの入力として参照できます。

4

テンプレートバージョンを に登録 AWS Proton し、テンプレートバンドルをアップロードします。

AWS Proton パラメータ

あなたの infrastructure as Code (IaC) ファイルにパラメータを定義して使用すると、ファイルの柔軟性が高まり、再利用可能になります。IaC ファイルでパラメータ値を読み取るには、パラメータ AWS Proton namespace のパラメータ名を参照します。 は、リソースのプロビジョニング中に生成されるレンダリングされた IaC ファイルにパラメータ値 AWS Proton を挿入します。 AWS CloudFormation IaC パラメータを処理するために、 は Jinja AWS Proton を使用します。Terraform IaC パラメータを処理するために、 は Terraform パラメータ値ファイル AWS Proton を生成し、HCL に組み込まれているパラメータ化機能に依存します。

ではCodeBuild プロビジョニング、コードがインポートできる入力ファイル AWS Proton を生成します。このファイルは、あなたのテンプレートのマニフェスト内のプロパティに応じて、JSON ファイルか HCL ファイルになります。詳細については、「the section called “CodeBuild プロビジョニングパラメータ”」を参照してください。

あなたの環境、サービス、コンポーネントの IaC ファイルまたはプロビジョニングコードのパラメータは、以下の要件で参照できます。

- 各パラメータ名の長さが 100 文字を超えてはなりません。
- パラメータの名前空間とリソース名の組み合わせの長さが、リソース名の文字制限を超えてはなりません。

AWS Proton これらのクォータを超えると、プロビジョニングは失敗します。

パラメータタイプ

以下のパラメータタイプは、 AWS Proton IaC ファイルで参照できます。

入力パラメータ

環境とサービスインスタンスでは、環境やサービステンプレートに関連付けたスキーマファイル内で定義した入力パラメータを受け取ることができます。リソースの IaC ファイルにあるリソースの入力パラメータを参照できます。コンポーネント IaC ファイルは、コンポーネントがアタッチされているサービスインスタンスの入力パラメータを参照できます。

AWS Proton は入力パラメータ名をスキーマファイルと照合し、IaC ファイルで参照されているパラメータと照合して、リソースのプロビジョニング中に仕様ファイルで指定した入力値を挿入します。

出力パラメータ

出力は、あなたのどの IaC ファイルでも定義できます。出力には、テンプレートがプロビジョニングするリソースの名前、ID、ARN の場合もあれば、テンプレートの入力の 1 つを渡す方法である場合もあります。これらの出力は、他のリソースの IaC ファイルで参照できます。

CloudFormation IaC ファイルでは、`Outputs:` ブロック内の出力パラメータを定義します。Terraform IaC ファイルでは、`output` ステートメントを使用して各出力パラメータを定義します。

リソースパラメータ

AWS Proton は自動的に AWS Proton リソースパラメータを作成します。これらのパラメータは、 AWS Proton リソースオブジェクトのプロパティを公開します。リソースパラメータの例は `environment.name` です。

IaC ファイルでの AWS Proton パラメータの使用

IaC ファイルでパラメータ値を読み取るには、パラメータ名前空間で AWS Proton パラメータの名前を参照します。 AWS CloudFormation IaC ファイルの場合、Jinja 構文を使用し、パラメータを中括弧と引用符のペアで囲みます。

以下の表は、サポートされている各テンプレート言語のリファレンス構文と例です。

テンプレート言語	構文	例:「VPC」という名前の環境入力
CloudFormation	`"{{ parameter-name }}"`	`"{{ environment.inputs.VPC }}"`

テンプレート言語	構文	例:「VPC」という名前の環境入力
Terraform	`var.`*`parameter-name`*	`var.environment.inputs.VPC` 生成された Terraform 変数定義

> **Note**
>
> あなたの IaC ファイル内で CloudFormation 動的パラメータを使用する場合、誤解による Jinja エラーを防ぐため、それらをエスケープする必要があります。詳細については、トラブルシューティング AWS Protonを参照してください。

次の表に、すべての AWS Proton リソースパラメータの名前空間名を示します。各テンプレートファイルタイプでは、パラメータ名前空間の異なるサブセットを使用できます。

テンプレートファイル	パラメータタイプ	パラメータ名	説明
環境	リソース	`environment.` **`name`**	環境名
	input	`environment.inputs.` *`input-name`*	スキーマ定義の環境入力
サービス	リソース	`environment.` **`name`** `environment.` **`account_id`**	環境名と AWS アカウント ID
	output	`environment.outputs.` *`output-name`*	環境 IaC ファイル出力
	リソース	`service.`**`branch_name`** `service.`**`name`** `service.`**`repository_connection_arn`**	サービス名とコードのリポジトリ

テンプレートファイル	パラメータタイプ	パラメータ名	説明
		service.**repository_id**	
	リソース	service_instance. **name**	サービスインスタンス名
	input	service_instance.inputs. *input-name*	スキーマ定義サービスインスタンス入力
	リソース	service_instance.components .default. **name**	アタッチされたデフォルトコンポーネント名
	output	service_instance.components .default.outputs. *output-name*	アタッチされたデフォルトコンポーネント IaC ファイル出力
パイプライン	リソース	service_instance.environment. **name** service_instance.environment. **account_id**	サービスインスタンスの環境名と AWS アカウント ID
	output	service_instance.environment.outputs. *output-name*	サービスインスタンス環境 IaC ファイル出力
	input	pipeline.inputs. *input-name*	スキーマ定義パイプライン入力
	リソース	service.**branch_name** service.**name** service.**repository_connection_arn** service.**repository_id**	サービス名とコードのリポジトリ

テンプレートファイル	パラメータタイプ	パラメータ名	説明
	input	`service_instance.inputs.` *input-name*	スキーマ定義サービスインスタンス入力
	collection	`{% for service_instance in` **`service_instances`** `%}...{% endfor %}`	ループスルーできるサービスインスタンスのコレクション
コンポーネント	リソース	`environment.` **`name`** `environment.` **`account_id`**	環境名と AWS アカウント アカウント ID
	output	`environment.outputs.` *output-name*	環境 IaC ファイル出力
	リソース	`service.`**`branch_name`** `service.`**`name`** `service.`**`repository_connection_arn`** `service.`**`repository_id`**	サービス名とコードのリポジトリ (アタッチされたコンポーネント)
	リソース	`service_instance.` **`name`**	サービスインスタンス名 (アタッチされたコンポーネント)
	input	`service_instance.inputs.` *input-name*	スキーマ定義のサービスインスタンス入力 (アタッチされたコンポーネント)
	リソース	`component.` **`name`**	コンポーネント名

詳細と例については、さまざまなリソースタイプとテンプレート言語の IaC テンプレートファイル内のパラメータに関するサブトピックを参照してください。

トピック

- 環境 CloudFormation IaC ファイルパラメータの詳細と例
- サービス CloudFormation IaC ファイルパラメータの詳細と例
- コンポーネント CloudFormation IaC ファイルパラメータの詳細と例
- CloudFormation IaC ファイルのパラメータフィルター
- CodeBuild プロビジョニングパラメータの詳細と例
- Terraform Infrastructure as Code (IaC) ファイルパラメータの詳細と例

環境 CloudFormation IaC ファイルパラメータの詳細と例

あなたの環境 infrastructure as code (IaC) ファイルでパラメータを定義し、参照することができます。 AWS Proton パラメータ、パラメータタイプ、パラメータ名前空間、および IaC ファイルでパラメータを使用する方法の詳細については、「」を参照してくださいthe section called “パラメータ”。

環境パラメータを定義する

環境 IaC ファイルの入力パラメータと出力パラメータの両方を定義できます。

- 入力パラメータ — あなたの スキーマファイルに環境入力パラメータを定義します。

  以下に挙げるのは、一般的なユースケースの入力パラメータの例です。

  - VPC CIDR 値
  - ロードバランサーの設定
  - データベース設定
  - ヘルスチェックのタイムアウト

  管理者は環境を作成するときに、入力パラメータの値を指定できます。

  - コンソールを使用して、 AWS Proton が提供するスキーマベースのフォームに入力します。
  - 値が含まれる仕様は CLI で指定します。
- 出力パラメータ — あなたの環境 IaC ファイルで環境出力を定義します。その後、他のリソースの IaC ファイルでこれらの出力を参照できます。

環境 IaC ファイルのパラメータ値を読み取ります。

環境 IaC ファイル内の環境に関連するパラメータは読み取ることができます。パラメータ値を読み取るには、 AWS Proton パラメータ名前空間でパラメータの名前を参照します。

- 入力パラメータ — `environment.inputs.`*`input-name`* を参照して環境入力値を読み取ります。
- リソースパラメータ – などの名前を参照して AWS Proton リソースパラメータを読み取りますenvironment.name。

> **Note**
>
> 他のリソースの出力パラメータは、環境 IaC ファイルでは利用できません。

環境とパラメータのあるサービス IaC ファイルの例

以下の例は、環境 IaC ファイルにおけるパラメータ定義と参照です。以下の例では、次に、環境 IaC ファイルで定義されている環境出力パラメータをサービス IaC ファイルで参照する方法を示します。

Example環境 CloudFormation IaC ファイル

この例では、以下の点に注意してください。

- `environment.inputs.` 名前空間は環境入力パラメータを参照します。
- Amazon EC2 Systems Manager (SSM) パラメータ `StoreInputValue` は、環境入力を連結します。
- `MyEnvParameterValue` 出力には、出力パラメータと同じ入力パラメータ連結が表示されます。さらに 3 つの出力パラメータによって、入力パラメータが個別に公開されます。
- さらに 6 つの出力パラメータで、環境がプロビジョニングするリソースが公開されます。

```
Resources:
  StoreInputValue:
    Type: AWS::SSM::Parameter
    Properties:
      Type: String
```

```
      Value: "{{ environment.inputs.my_sample_input }}
 {{ environment.inputs.my_other_sample_input}}
 {{ environment.inputs.another_optional_input }}"
              # input parameter references

# These output values are available to service infrastructure as code files as outputs,
 when given the
# the 'environment.outputs' namespace, for example,
 service_instance.environment.outputs.ClusterName.
Outputs:
  MyEnvParameterValue:                                    # output definition
    Value: !GetAtt StoreInputValue.Value
  MySampleInputValue:                                     # output definition
    Value: "{{ environment.inputs.my_sample_input }}"         #   input parameter
 reference
  MyOtherSampleInputValue:                                # output definition
    Value: "{{ environment.inputs.my_other_sample_input }}"   #   input parameter
 reference
  AnotherOptionalInputValue:                              # output definition
    Value: "{{ environment.inputs.another_optional_input }}"  #   input parameter
 reference
  ClusterName:                                            # output definition
    Description: The name of the ECS cluster
    Value: !Ref 'ECSCluster'                              #   provisioned resource
  ECSTaskExecutionRole:                                   # output definition
    Description: The ARN of the ECS role
    Value: !GetAtt 'ECSTaskExecutionRole.Arn'             #   provisioned resource
  VpcId:                                                  # output definition
    Description: The ID of the VPC that this stack is deployed in
    Value: !Ref 'VPC'                                     #   provisioned resource
  PublicSubnetOne:                                        # output definition
    Description: Public subnet one
    Value: !Ref 'PublicSubnetOne'                         #   provisioned resource
  PublicSubnetTwo:                                        # output definition
    Description: Public subnet two
    Value: !Ref 'PublicSubnetTwo'                         #   provisioned resource
  ContainerSecurityGroup:                                 # output definition
    Description: A security group used to allow Fargate containers to receive traffic
    Value: !Ref 'ContainerSecurityGroup'                  #   provisioned resource
```

Exampleサービス CloudFormation IaC ファイル

environment.outputs. 名前空間は、環境 IaC ファイルからの環境出力を参照します。たとえば、名前 environment.outputs.ClusterName は ClusterName 環境出力パラメータの値を読み取ります。

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Deploy a service on AWS Fargate, hosted in a public subnet, and accessible
 via a public load balancer.
Mappings:
  TaskSize:
    x-small:
      cpu: 256
      memory: 512
    small:
      cpu: 512
      memory: 1024
    medium:
      cpu: 1024
      memory: 2048
    large:
      cpu: 2048
      memory: 4096
    x-large:
      cpu: 4096
      memory: 8192
Resources:
  # A log group for storing the stdout logs from this service's containers
  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: '{{service_instance.name}}' # resource parameter

  # The task definition. This is a simple metadata description of what
  # container to run, and what resource requirements it has.
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      Family: '{{service_instance.name}}' # resource parameter
      Cpu: !FindInMap [TaskSize, {{service_instance.inputs.task_size}}, cpu] # input
 parameter
      Memory: !FindInMap [TaskSize, {{service_instance.inputs.task_size}}, memory]
      NetworkMode: awsvpc
```

```
      RequiresCompatibilities:
        - FARGATE
      ExecutionRoleArn: '{{environment.outputs.ECSTaskExecutionRole}}'  # output
reference to an environment infrastructure code file
      TaskRoleArn: !Ref "AWS::NoValue"
      ContainerDefinitions:
        - Name: '{{service_instance.name}}'  # resource parameter
          Cpu: !FindInMap [TaskSize, {{service_instance.inputs.task_size}}, cpu]
          Memory: !FindInMap [TaskSize, {{service_instance.inputs.task_size}}, memory]
          Image: '{{service_instance.inputs.image}}'
          PortMappings:
            - ContainerPort: '{{service_instance.inputs.port}}' # input parameter
          LogConfiguration:
            LogDriver: 'awslogs'
            Options:
              awslogs-group: '{{service_instance.name}}' # resource parameter
              awslogs-region: !Ref 'AWS::Region'
              awslogs-stream-prefix: '{{service_instance.name}}' # resource parameter

  # The service_instance. The service is a resource which allows you to run multiple
  # copies of a type of task, and gather up their logs and metrics, as well
  # as monitor the number of running tasks and replace any that have crashed
  Service:
    Type: AWS::ECS::Service
    DependsOn: LoadBalancerRule
    Properties:
      ServiceName: '{{service_instance.name}}'  # resource parameter
      Cluster: '{{environment.outputs.ClusterName}}' # output reference to an
environment infrastructure as code file
      LaunchType: FARGATE
      DeploymentConfiguration:
        MaximumPercent: 200
        MinimumHealthyPercent: 75
      DesiredCount: '{{service_instance.inputs.desired_count}}'# input parameter
      NetworkConfiguration:
        AwsvpcConfiguration:
          AssignPublicIp: ENABLED
          SecurityGroups:
            - '{{environment.outputs.ContainerSecurityGroup}}' # output reference to an
environment infrastructure as code file
          Subnets:
            - '{{environment.outputs.PublicSubnetOne}}' # output reference to an
environment infrastructure as code file
```

```
            - '{{environment.outputs.PublicSubnetTwo}}' # output reference to an
 environment infrastructure as code file
      TaskDefinition: !Ref 'TaskDefinition'
      LoadBalancers:
        - ContainerName: '{{service_instance.name}}'  # resource parameter
          ContainerPort: '{{service_instance.inputs.port}}' # input parameter
          TargetGroupArn: !Ref 'TargetGroup'
[...]
```

サービス CloudFormation IaC ファイルパラメータの詳細と例

パラメータは、あなたのサービスとパイプラインの Infrastructure as Code (IaC)ファイルで定義し、参照することができます。パラメータ、パラメータタイプ、 AWS Proton パラメータ名前空間、およびあなたの IaC ファイル内のパラメータの使用方法の詳細については、the section called “パラメータ” を参照してください。

サービスパラメータを定義する

サービス IaC ファイルには、入力パラメータと出力パラメータの両方を定義できます。

- 入力パラメータ — サービスインスタンスの入力パラメータをあなたのスキーマファイルに定義します。

  以下に挙げるのは、一般的なユースケースのサービス入力パラメータの例です。

  - ポート
  - タスクサイズ
  - イメージ
  - 必要数
  - Docker ファイル
  - ユニットテストコマンド

  サービスを作成するときに、入力パラメータの値を指定できます。

  - コンソールを使用して、 AWS Proton が提供するスキーマベースのフォームに入力します。
  - 値を含む仕様を CLI で指定します。
- 出力パラメータ — あなたのサービス IaC ファイル内のサービスインスタンス出力を定義します。その後、他のリソースの IaC ファイルでこれらの出力を参照できます。

サービス IaC ファイル内のパラメータ値を読み取ります。

サービス IaC ファイル内のサービスや他のリソースに関連するパラメータを読み取ることができます。パラメータ値を読み取るには、パラメータ名前空間で AWS Proton パラメータの名前を参照します。

- 入力パラメータ — `service_instance.inputs.`*`input-name`* を参照してサービスインスタンスの入力値を読み込む。
- リソースパラメータ – `service.name`、、 などの名前を参照して`service_instance.name`リソース AWS Proton パラメータを読み取ります`environment.name`。
- 出力パラメータ — `environment.outputs.`*`output-name`* または `service_instance.components.default.outputs.`*`output-name`* を参照して他のリソースの出力を読み取ります。

パラメータのあるサービス IaC ファイルの例

次の例は、サービス CloudFormation IaC ファイルからのスニペットです。`environment.outputs.` 名前空間は、環境 IaC ファイルからの出力を参照します。`service_instance.inputs.` 名前空間はサービスインスタンスの入力パラメータを参照します。`service_instance.name` プロパティは、 AWS Proton リソースパラメータを参照します。

```
Resources:
  StoreServiceInstanceInputValue:
    Type: AWS::SSM::Parameter
    Properties:
      Type: String
      Value: "{{ service.name }} {{ service_instance.name }}
 {{ service_instance.inputs.my_sample_service_instance_required_input }}
 {{ service_instance.inputs.my_sample_service_instance_optional_input }}
 {{ environment.outputs.MySampleInputValue }}
 {{ environment.outputs.MyOtherSampleInputValue }}"
              #  resource parameter references               # input parameter
 references
                                          # output references to an environment
 infrastructure as code file
Outputs:
  MyServiceInstanceParameter:                                                #
 output definition
    Value: !Ref StoreServiceInstanceInputValue
```

```
 MyServiceInstanceRequiredInputValue:                                    #
output definition
   Value: "{{ service_instance.inputs.my_sample_service_instance_required_input }}"  #
input parameter reference
 MyServiceInstanceOptionalInputValue:                                    #
output definition
   Value: "{{ service_instance.inputs.my_sample_service_instance_optional_input }}"  #
input parameter reference
 MyServiceInstancesEnvironmentSampleOutputValue:                         #
output definition
   Value: "{{ environment.outputs.MySampleInputValue }}"                 #
output reference to an environment IaC file
 MyServiceInstancesEnvironmentOtherSampleOutputValue:                    #
output definition
   Value: "{{ environment.outputs.MyOtherSampleInputValue }}"            #
output reference to an environment IaC file
```

コンポーネント CloudFormation IaC ファイルパラメータの詳細と例

あなたのコンポーネント infrastructure as code (IaC)ファイルで、パラメータを定義して参照することができます。 AWS Proton パラメータ、パラメータタイプ、パラメータ名前空間、および IaC ファイルでパラメータを使用する方法の詳細については、「」を参照してくださいthe section called "パラメータ"。コンポーネントの詳細については、「コンポーネント」を参照してください。

コンポーネントの出力パラメータを定義します。

出力パラメータは、あなたのコンポーネント IaC ファイルで定義できます。その後、これらの出力をサービス IaC ファイルで参照できます。

> **Note**
>
> コンポーネント IaC ファイルには入力を定義できません。アタッチされたコンポーネントは、アタッチ先のサービスインスタンスから入力を取得できます。デタッチされたコンポーネントには入力がありません。

コンポーネント IaC ファイルのパラメータ値を読み取る

コンポーネント IaC ファイル内のコンポーネントや他のリソースに関連するパラメータを読み取ることができます。パラメータ値を読み取るには、パラメータ名前空間で AWS Proton パラメータの名前を参照します。

- 入力パラメータ — `service_instance.inputs.input-name` を参照して、アタッチされたサービスインスタンス入力値を読み込みます。
- リソースパラメータ – 、`component.name`、`service.name`、 などの名前を参照して`service_instance.name`リソース AWS Proton パラメータを読み取りますす`environment.name`。
- 出力パラメータ — `environment.outputs.output-name` を参照して環境出力を読み取ります。

パラメータを含むコンポーネントとサービス IaC ファイルの例

次の例では、Amazon Simple Storage Service (Amazon S3) バケットと関連するアクセスポリシーをプロビジョニングし、両方のリソースの Amazon リソースネーム (ARN) をコンポーネント出力として公開するコンポーネントを示します。コンポーネント出力をコンテナで実行しているコードで利用できるように、サービス IaC テンプレートは、Amazon Elastic Container Service (Amazon ECS) タスクのコンテナ環境変数として追加し、バケットアクセスポリシーをタスクのロールに追加します。バケット名は環境、サービス、サービスインスタンス、およびコンポーネントの名前に基づいています。言い換えると、バケットはコンポーネントテンプレートの特定のインスタンスと結合して特定のサービスインスタンスを拡張します。開発者は、このコンポーネントテンプレートに基づいて複数のカスタムコンポーネントを作成し、さまざまなサービスインスタンスや機能上のニーズに合わせて Amazon S3 バケットをプロビジョニングできます。

この例では、Jinja `{{ ... }}` 構文で、あなたのサービス IaC ファイル内のコンポーネントやその他のリソースパラメータを参照する方法を示します。`{% if ... %}` ステートメントでステートメントのブロックを追加できるのは、コンポーネントがサービスインスタンスにアタッチされている場合のみです。`proton_cfn_*` キーワードは、出力パラメータ値のサニタイズやフォーマットに使用できるフィルターです。フィルターの詳細については、「the section called “CloudFormation パラメータフィルター”」を参照してください。

管理者としてのあなたは、サービス IaC テンプレートファイルをオーサリングします。

Exampleコンポーネントで CloudFormation IaC ファイルをサービスする

```
# service/instance_infrastructure/cloudformation.yaml

Resources:
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      TaskRoleArn: !Ref TaskRole
```

```
      ContainerDefinitions:
        - Name: '{{service_instance.name}}'
          # ...
          {% if service_instance.components.default.outputs | length > 0 %}
          Environment:
            {{ service_instance.components.default.outputs |
                proton_cfn_ecs_task_definition_formatted_env_vars }}
          {% endif %}

  # ...

  TaskRole:
    Type: AWS::IAM::Role
    Properties:
      # ...
      ManagedPolicyArns:
        - !Ref BaseTaskRoleManagedPolicy
        {{ service_instance.components.default.outputs
            | proton_cfn_iam_policy_arns }}

  # Basic permissions for the task
  BaseTaskRoleManagedPolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      # ...
```

開発者としてのあなたは、コンポーネント IaC テンプレートファイルをオーサリングします。

Exampleコンポーネント CloudFormation IaC ファイル

```
# cloudformation.yaml

# A component that defines an S3 bucket and a policy for accessing the bucket.
Resources:
  S3Bucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: '{{environment.name}}-{{service.name}}-{{service_instance.name}}-
{{component.name}}'
  S3BucketAccessPolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      PolicyDocument:
        Version: "2012-10-17"
```

```
        Statement:
          - Effect: Allow
            Action:
              - 's3:Get*'
              - 's3:List*'
              - 's3:PutObject'
            Resource: !GetAtt S3Bucket.Arn
Outputs:
  BucketName:
    Description: "Bucket to access"
    Value: !GetAtt S3Bucket.Arn
  BucketAccessPolicyArn:
    Value: !Ref S3BucketAccessPolicy
```

がサービスインスタンスの CloudFormation テンプレートを AWS Proton レンダリングし、すべてのパラメータを実際の値に置き換えると、テンプレートは次のファイルのようになります。

Exampleサービスインスタンス CloudFormation でレンダリングした IaC ファイル

```
Resources:
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      TaskRoleArn: !Ref TaskRole
      ContainerDefinitions:
        - Name: '{{service_instance.name}}'
          # ...
          Environment:
            - Name: BucketName
              Value: arn:aws:s3:us-
east-1:123456789012:environment_name-service_name-service_instance_name-component_name
            - Name: BucketAccessPolicyArn
              Value: arn:aws:iam::123456789012:policy/cfn-generated-policy-name
  # ...

  TaskRole:
    Type: AWS::IAM::Role
    Properties:
      # ...
      ManagedPolicyArns:
        - !Ref BaseTaskRoleManagedPolicy
        - arn:aws:iam::123456789012:policy/cfn-generated-policy-name
```

```
  # Basic permissions for the task
  BaseTaskRoleManagedPolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      # ...
```

CloudFormation IaC ファイルのパラメータフィルター

AWS CloudFormation IaC ファイル内のAWS Proton パラメータを参照するときは、フィルターと呼ばれる Jinja 修飾子を使用して、レンダリングされたテンプレートに挿入される前にパラメータ値を検証、フィルタリング、フォーマットできます。コンポーネントの作成とアタッチは開発者が行いますが、サービスインスタンステンプレートでコンポーネント出力を使用する管理者はその存在と有効性を検証することがあるため、フィルター検証機能はコンポーネント出力パラメータを参照するときに特に便利です。ただし、フィルターは、どの Jinja IaC ファイルでも使用できます。

以下のセクションでは、使用可能なパラメータフィルターについて説明および定義し、これらのフィルターのほとんどを examples. AWS Proton defines で定義します。`default` フィルターは Jinja の組み込みフィルターです。

Amazon ECS タスク用の環境プロパティをフォーマットする

宣言

```
dict # proton_cfn_ecs_task_definition_formatted_env_vars (raw: boolean = True) # YAML
 list of dicts
```

説明

このフィルターは、Amazon Elastic Container Service (Amazon ECS) タスク定義の `ContainerDefinition` セクションにある環境プロパティで使用される出力のリストをフォーマットします。

`raw` を `False` に設定すると、パラメータ値も検証されます。この場合、値は正規表現 `^[a-zA-Z0-9_-]*$` と一致させる必要があります。値がこの検証に失敗すると、テンプレートのレンダリングは失敗します。

例

以下のカスタムコンポーネントテンプレートを使用します。

```
Resources:
```

```
  # ...
Outputs:
  Output1:
    Description: "Example component output 1"
    Value: hello
  Output2:
    Description: "Example component output 2"
    Value: world
```

そして、以下のサービステンプレート:

```
Resources:
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      # ...
      ContainerDefinitions:
        - Name: MyServiceName
          # ...
          Environment:
            {{ service_instance.components.default.outputs
              | proton_cfn_ecs_task_definition_formatted_env_vars }}
```

レンダリングされたサービステンプレートは以下のとおりです。

```
Resources:
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      # ...
      ContainerDefinitions:
        - Name: MyServiceName
          # ...
          Environment:
            - Name: Output1
              Value: hello
            - Name: Output2
              Value: world
```

Lambda 関数のフォーマット環境プロパティ

宣言

```
dict # proton_cfn_lambda_function_formatted_env_vars (raw: boolean = True) # YAML dict
```

説明

このフィルターは、 AWS Lambda 関数定義の Propertiesセクションの環境プロパティで使用される出力のリストをフォーマットします。

raw を False に設定すると、パラメータ値も検証されます。この場合、値は正規表現 ^[a-zA-Z0-9_-]*$ と一致させる必要があります。値がこの検証に失敗すると、テンプレートのレンダリングは失敗します。

例

以下のカスタムコンポーネントテンプレートを使用します。

```
Resources:
  # ...
Outputs:
  Output1:
    Description: "Example component output 1"
    Value: hello
  Output2:
    Description: "Example component output 2"
    Value: world
```

そして、以下のサービステンプレート:

```
Resources:
  Lambda:
    Type: AWS::Lambda::Function
    Properties:
      Environment:
        Variables:
          {{ service_instance.components.default.outputs
            | proton_cfn_lambda_function_formatted_env_vars }}
```

レンダリングされたサービステンプレートは以下のとおりです。

```
Resources:
  Lambda:
    Type: AWS::Lambda::Function
    Properties:
```

```
      Environment:
        Variables:
          Output1: hello
          Output2: world
```

IAM ポリシー ARN を抽出して IAM ロールに含める

宣言

```
dict # proton_cfn_iam_policy_arns # YAML list
```

説明

このフィルターは、 AWS Identity and Access Management (IAM) ロール定義の `Properties`セクションの ManagedPolicyArns プロパティで使用される出力のリストをフォーマットします。フィルターは正規表現を使用して、`^arn:[a-zA-Z-]+:iam::\d{12}:policy/` 出力パラメータのリストから有効な IAM ポリシー ARN を抽出します。このフィルターを使用して、出力パラメータ値のポリシーをサービステンプレートの IAM ロール定義に追加できます。

例

以下のカスタムコンポーネントテンプレートを使用します。

```
Resources:
  # ...
  ExamplePolicy1:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      # ...
  ExamplePolicy2:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      # ...

  # ...

Outputs:
  Output1:
    Description: "Example component output 1"
    Value: hello
  Output2:
    Description: "Example component output 2"
    Value: world
```

```
  PolicyArn1:
    Description: "ARN of policy 1"
    Value: !Ref ExamplePolicy1
  PolicyArn2:
    Description: "ARN of policy 2"
    Value: !Ref ExamplePolicy2
```

そして、以下のサービステンプレート:

```
Resources:

  # ...

  TaskRole:
    Type: AWS::IAM::Role
    Properties:
      # ...
      ManagedPolicyArns:
        - !Ref BaseTaskRoleManagedPolicy
        {{ service_instance.components.default.outputs
            | proton_cfn_iam_policy_arns }}

  # Basic permissions for the task
  BaseTaskRoleManagedPolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      # ...
```

レンダリングされたサービステンプレートは以下のとおりです。

```
Resources:

  # ...

  TaskRole:
    Type: AWS::IAM::Role
    Properties:
      # ...
      ManagedPolicyArns:
        - !Ref BaseTaskRoleManagedPolicy
        - arn:aws:iam::123456789012:policy/cfn-generated-policy-name-1
        - arn:aws:iam::123456789012:policy/cfn-generated-policy-name-2
```

```
  # Basic permissions for the task
  BaseTaskRoleManagedPolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      # ...
```

プロパティ値をサニタイズする

宣言

```
string # proton_cfn_sanitize # string
```

説明

これは汎用フィルターです。これを使用してパラメータ値の安全性を検証します。フィルターは、値が正規表現 ^[a-zA-Z0-9_-]*$ と一致するか、有効な Amazon リソースネーム (ARN) であるかを検証します。値がこの検証に合格しなければ、テンプレートのレンダリングは失敗します。

例

以下のカスタムコンポーネントテンプレートを使用:

```
Resources:
  # ...
Outputs:
  Output1:
    Description: "Example of valid output"
    Value: "This-is_valid_37"
  Output2:
    Description: "Example incorrect output"
    Value: "this::is::incorrect"
  SomeArn:
    Description: "Example ARN"
    Value: arn:aws:some-service::123456789012:some-resource/resource-name
```

- サービステンプレート内の以下のリファレンス:

  ```
  # ...
    {{ service_instance.components.default.outputs.Output1
      | proton_cfn_sanitize }}
  ```

  以下のようにレンダリングされます:

```
# ...
  This-is_valid_37
```

- サービステンプレート内の以下のリファレンス:

```
# ...
  {{ service_instance.components.default.outputs.Output2
    | proton_cfn_sanitize }}
```

その結果、次のレンダリングエラーが発生します。

```
Illegal character(s) detected in "this::is::incorrect". Must match regex ^[a-zA-
Z0-9_-]*$ or be a valid ARN
```

- サービステンプレート内の以下のリファレンス:

```
# ...
  {{ service_instance.components.default.outputs.SomeArn
    | proton_cfn_sanitize }}
```

以下のようにレンダリングされます。

```
# ...
  arn:aws:some-service::123456789012:some-resource/resource-name
```

存在しない参照にはデフォルト値を指定します。

説明

名前空間参照が存在しない場合、default フィルターはデフォルト値を提供します。デフォルト値を使えば、参照するパラメータが欠落していても問題なくレンダリングできる堅牢なテンプレートを作成できます。

例

サービスインスタンスに直接定義された (デフォルト) コンポーネントがアタッチされていない場合や、アタッチされたコンポーネントに test という名前の出力がない場合、サービステンプレート内の次の参照によってテンプレートのレンダリングが失敗します。

```
# ...
  {{ service_instance.components.default.outputs.test }}
```

この問題を回避するには、default フィルターを追加してください。

```
# ...
  {{ service_instance.components.default.outputs.test | default("[optional-value]") }}
```

CodeBuild プロビジョニングパラメータの詳細と例

CodeBuild ベースの AWS Proton リソースのテンプレートでパラメータを定義し、プロビジョニングコードでこれらのパラメータを参照できます。 AWS Proton パラメータ、パラメータタイプ、パラメータ名前空間、および IaC ファイルでパラメータを使用する方法の詳細については、「」を参照してくださいthe section called “パラメータ”。

> **Note**
>
> CodeBuild プロビジョニングは環境とサービスで使用できます。現時点では、この方法でコンポーネントをプロビジョニングすることはできません。

入力パラメータ

環境やサービスなどの AWS Proton リソースを作成するときは、テンプレートのスキーマファイルで定義されている入力パラメータの値を指定します。作成したリソースが を使用する場合CodeBuild プロビジョニング、 はこれらの入力値を入力ファイルに AWS Proton レンダリングします。あなたのプロビジョニングコードでは、このファイルからパラメータ値をインポートして取得できます。

CodeBuild テンプレートの例については、「the section called “CodeBuild バンドル”」を参照してください。マニフェストファイルについて詳しくは、「the section called “マニフェストとまとめ”」を参照してください。

次の例は、CodeBuild ベースのサービスインスタンスのプロビジョニング中に生成される JSON 入力ファイルです。

例: CodeBuild プロビジョニング AWS CDK で を使用する

```
{
  "service_instance": {
```

```
    "name": "my-service-staging",
    "inputs": {
      "port": "8080",
      "task_size": "medium"
    }
  },
  "service": {
    "name": "my-service"
  },
  "environment": {
    "account_id": "123456789012",
    "name": "my-env-staging",
    "outputs": {
      "vpc-id": "hdh2323423"
    }
  }
}
```

出力パラメータ

リソースプロビジョニング出力を に伝達するために AWS Proton、プロビジョニングコードは、テンプレートのスキーマファイルで定義された出力パラメータの値proton-outputs.jsonを持つ という名前の JSON ファイルを生成できます。 ???たとえば、 cdk deploy コマンドには、プロビジョニング出力を含む JSON ファイルを生成する AWS CDK ように に指示する --outputs-file引数があります。リソースが を使用している場合は AWS CDK、CodeBuild テンプレートマニフェストで次のコマンドを指定します。

```
aws proton notify-resource-deployment-status-change
```

AWS Proton はこの JSON ファイルを検索します。プロビジョニングコードが正常に完了した後にファイルが存在する場合、 はそのファイルから出力パラメータ値を AWS Proton 読み取ります。

Terraform Infrastructure as Code (IaC) ファイルパラメータの詳細と例

Terraform の入力変数は、あなたのテンプレートバンドル内の variable.tf ファイルに含めることができます。スキーマを作成して、スキーマファイルから管理変数を作成 AWS Proton することもできます。 AWS Proton は、スキーマファイル.tf filesから変数を作成します。詳細については、「the section called “Terraform IaC ファイル”」を参照してください。

インフラストラクチャ でスキーマ定義 AWS Proton 変数を参照するには.tf files、Terraform IaC のパラメータと AWS Proton 名前空間 テーブルに示されている名前空間を使用しま

す。 IaC たとえば、var.environment.inputs.vpc_cidr を使用できます。引用符の中で、これらの変数を単一角括弧で囲み、先頭の中括弧の前にドル記号を追加します (たとえば、“${var.environment.inputs.vpc_cidr}”)。

次の例は、名前空間を使用して環境 に AWS Proton パラメータを含める方法を示しています.tf file。

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
  // This tells terraform to store the state file in s3 at the location
  // s3://terraform-state-bucket/tf-os-sample/terraform.tfstate
  backend "s3" {
    bucket = "terraform-state-bucket"
    key    = "tf-os-sample/terraform.tfstate"
    region = "us-east-1"
  }
}

// Configure the AWS Provider
provider "aws" {
  region = "us-east-1"
  default_tags {
    tags = var.proton_tags
  }
}

resource "aws_ssm_parameter" "my_ssm_parameter" {
  name  = "my_ssm_parameter"
  type  = "String"
  // Use the Proton environment.inputs. namespace
  value = var.environment.inputs.ssm_parameter_value
}
```

AWS Proton コードファイルとしての インフラストラクチャ

テンプレートバンドルの主な部分は、プロビジョニングするインフラストラクチャリソースとプロパティを定義する infrastructure as code (IaC) ファイルです。 AWS CloudFormation また、コードエン

ジンはこれらのタイプのファイルを使用してインフラストラクチャリソースをプロビジョニングします。

> **Note**
>
> IaC ファイルは、直接定義されたコンポーネントへの直接入力として、テンプレートバンドルとは関係なく使用することもできます。コンポーネントの詳細については、「コンポーネント」を参照してください。

AWS Proton は現在、次の 2 種類の IaC ファイルをサポートしています。

- CloudFormation ファイル — AWSマネージドプロビジョニングに使用します。 AWS Proton では CloudFormation テンプレートファイルフォーマットの上に Jinja でパラメータを設定します。
- Terraform HCL ファイル — セルフマネージドのプロビジョニングに使用します。HCL はパラメータ化をネイティブにサポートします。

プロビジョニング方法の組み合わせを使用して AWS Proton リソースをプロビジョニングすることはできません。いずれか 1 つを使用する必要があります。 AWSマネージドプロビジョニングサービスをセルフマネージドプロビジョニング環境にデプロイしたり、その逆を実行したりすることはできません。

詳細については、「the section called “プロビジョニングの方法”」、「環境」、「サービス」、および「コンポーネント」を参照してください。

CloudFormation IaC ファイル

で AWS CloudFormation infrastructure as code ファイルを使用する方法について説明します AWS Proton。 CloudFormation は、 AWS リソースのモデル化とセットアップに役立つ infrastructure as code (IaC) サービスです。テンプレートでインフラストラクチャリソースを定義します。パラメータ化用の CloudFormation テンプレートファイル形式の上に Jinja を使用します。 はパラメータ AWS Proton を展開し、CloudFormation テンプレート全体をレンダリングします。CloudFormation は、定義されたリソースを CloudFormation スタックとしてプロビジョニングします。詳細については、 『CloudFormation ユーザーガイド』の「CloudFormationとは?」を参照してください。

AWS Proton は CloudFormation IaC の AWSマネージドプロビジョニングをサポートしています。

既存の独自の Infrastructure as Code (IaC) ファイルから開始する

独自の既存の Infrastructure as Code (IaC) ファイルを適応させて使用できます AWS Proton。

次の CloudFormation 例、例 1、例 2 は、独自の既存の CloudFormation IaC ファイルを表します。CloudFormation では、これらのファイルで 2 つの異なる CloudFormation スタックを作成できます。

例 1 では、CloudFormation IaC ファイルは、コンテナアプリケーション間で共有されるインフラストラクチャをプロビジョニングするように設定されています。この例では、同じ IaC ファイルでプロビジョニングされたインフラストラクチャの複数のセットを作成できるように、入力パラメータを追加しています。各セットには、異なる名前と、VPC およびサブネット CIDR 値のセットを指定できます。IaC ファイルを使用して CloudFormation でインフラストラクチャリソースをプロビジョニングするときに、管理者または開発者として、これらのパラメータの値を指定します。便宜上、これらの入力パラメータはコメント行のマークが付いており、例では複数回参照されています。出力は、テンプレートの終わりで定義されます。これらは、他の CloudFormation IaC ファイル内で参照できます。

例 2 では、CloudFormation IaC ファイルは、例 1 でプロビジョニングされたインフラストラクチャにアプリケーションをデプロイするように設定されています。便宜上、パラメータはコメント化されています。

例 1: CloudFormation IaC ファイル

```
AWSTemplateFormatVersion: '2010-09-09'
Description: AWS Fargate cluster running containers in a public subnet. Only supports
             public facing load balancer, and public service discovery namespaces.
Parameters:
   VpcCIDR:       # input parameter
        Description: CIDR for VPC
        Type: String
        Default: "10.0.0.0/16"
   SubnetOneCIDR: # input parameter
        Description: CIDR for SubnetOne
        Type: String
        Default: "10.0.0.0/24"
   SubnetTwoCIDR: # input parameters
        Description: CIDR for SubnetTwo
        Type: String
        Default: "10.0.1.0/24"
Resources:
  VPC:
```

```
    Type: AWS::EC2::VPC
    Properties:
      EnableDnsSupport: true
      EnableDnsHostnames: true
      CidrBlock:
        Ref: 'VpcCIDR'

  # Two public subnets, where containers will have public IP addresses
  PublicSubnetOne:
    Type: AWS::EC2::Subnet
    Properties:
      AvailabilityZone:
         Fn::Select:
         - 0
         - Fn::GetAZs: {Ref: 'AWS::Region'}
      VpcId: !Ref 'VPC'
      CidrBlock:
         Ref: 'SubnetOneCIDR'
      MapPublicIpOnLaunch: true

  PublicSubnetTwo:
    Type: AWS::EC2::Subnet
    Properties:
      AvailabilityZone:
         Fn::Select:
         - 1
         - Fn::GetAZs: {Ref: 'AWS::Region'}
      VpcId: !Ref 'VPC'
      CidrBlock:
        Ref: 'SubnetTwoCIDR'
      MapPublicIpOnLaunch: true

  # Setup networking resources for the public subnets. Containers
  # in the public subnets have public IP addresses and the routing table
  # sends network traffic via the internet gateway.
  InternetGateway:
    Type: AWS::EC2::InternetGateway
  GatewayAttachement:
    Type: AWS::EC2::VPCGatewayAttachment
    Properties:
      VpcId: !Ref 'VPC'
      InternetGatewayId: !Ref 'InternetGateway'
  PublicRouteTable:
    Type: AWS::EC2::RouteTable
```

```
    Properties:
      VpcId: !Ref 'VPC'
  PublicRoute:
    Type: AWS::EC2::Route
    DependsOn: GatewayAttachement
    Properties:
      RouteTableId: !Ref 'PublicRouteTable'
      DestinationCidrBlock: '0.0.0.0/0'
      GatewayId: !Ref 'InternetGateway'
  PublicSubnetOneRouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      SubnetId: !Ref PublicSubnetOne
      RouteTableId: !Ref PublicRouteTable
  PublicSubnetTwoRouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      SubnetId: !Ref PublicSubnetTwo
      RouteTableId: !Ref PublicRouteTable

  # ECS Resources
  ECSCluster:
    Type: AWS::ECS::Cluster

  # A security group for the containers we will run in Fargate.
  # Rules are added to this security group based on what ingress you
  # add for the cluster.
  ContainerSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Access to the Fargate containers
      VpcId: !Ref 'VPC'

  # This is a role which is used by the ECS tasks themselves.
  ECSTaskExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
        - Effect: Allow
          Principal:
            Service: [ecs-tasks.amazonaws.com]
          Action: ['sts:AssumeRole']
      Path: /
```

```
      ManagedPolicyArns:
        - 'arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy'

# These output values will be available to other templates to use.
Outputs:
  ClusterName:                                              # output
    Description: The name of the ECS cluster
    Value: !Ref 'ECSCluster'
    Export:
      Name:
        Fn::Sub: "${AWS::StackName}-ECSCluster"
  ECSTaskExecutionRole:                                      # output
    Description: The ARN of the ECS role
    Value: !GetAtt 'ECSTaskExecutionRole.Arn'
    Export:
      Name:
        Fn::Sub: "${AWS::StackName}-ECSTaskExecutionRole"
  VpcId:                                                     # output
    Description: The ID of the VPC that this stack is deployed in
    Value: !Ref 'VPC'
    Export:
      Name:
        Fn::Sub: "${AWS::StackName}-VPC"
  PublicSubnetOne:                                           # output
    Description: Public subnet one
    Value: !Ref 'PublicSubnetOne'
    Export:
      Name:
        Fn::Sub: "${AWS::StackName}-PublicSubnetOne"
  PublicSubnetTwo:                                           # output
    Description: Public subnet two
    Value: !Ref 'PublicSubnetTwo'
    Export:
      Name:
        Fn::Sub: "${AWS::StackName}-PublicSubnetTwo"
  ContainerSecurityGroup:                                    # output
    Description: A security group used to allow Fargate containers to receive traffic
    Value: !Ref 'ContainerSecurityGroup'
    Export:
      Name:
        Fn::Sub: "${AWS::StackName}-ContainerSecurityGroup"
```

例 2: CloudFormation IaC ファイル

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Deploy a service on AWS Fargate, hosted in a public subnet, and accessible
 via a public load balancer.
Parameters:
    ContainerPortInput:  # input parameter
        Description: The port to route traffic to
        Type: Number
        Default: 80
    TaskCountInput:      # input parameter
        Description: The default number of Fargate tasks you want running
        Type: Number
        Default: 1
    TaskSizeInput:       # input parameter
        Description: The size of the task you want to run
        Type: String
        Default: x-small
    ContainerImageInput: # input parameter
        Description: The name/url of the container image
        Type: String
        Default: "public.ecr.aws/z9d2n7e1/nginx:1.19.5"
    TaskNameInput:       # input parameter
        Description: Name for your task
        Type: String
        Default: "my-fargate-instance"
    StackName:           # input parameter
        Description: Name of the environment stack to deploy to
        Type: String
        Default: "my-fargate-environment"
Mappings:
  TaskSizeMap:
    x-small:
      cpu: 256
      memory: 512
    small:
      cpu: 512
      memory: 1024
    medium:
      cpu: 1024
      memory: 2048
    large:
      cpu: 2048
      memory: 4096
```

```
    x-large:
      cpu: 4096
      memory: 8192
Resources:
  # A log group for storing the stdout logs from this service's containers
  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName:
        Ref: 'TaskNameInput' # input parameter

  # The task definition. This is a simple metadata description of what
  # container to run, and what resource requirements it has.
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      Family: !Ref 'TaskNameInput'
      Cpu: !FindInMap [TaskSizeMap, !Ref 'TaskSizeInput', cpu]
      Memory: !FindInMap [TaskSizeMap, !Ref 'TaskSizeInput', memory]
      NetworkMode: awsvpc
      RequiresCompatibilities:
        - FARGATE
      ExecutionRoleArn:
        Fn::ImportValue:
          !Sub "${StackName}-ECSTaskExecutionRole"    # output parameter from another
 CloudFormation template
              awslogs-region: !Ref 'AWS::Region'
              awslogs-stream-prefix: !Ref 'TaskNameInput'


  # The service_instance. The service is a resource which allows you to run multiple
  # copies of a type of task, and gather up their logs and metrics, as well
  # as monitor the number of running tasks and replace any that have crashed
  Service:
    Type: AWS::ECS::Service
    DependsOn: LoadBalancerRule
    Properties:
      ServiceName: !Ref 'TaskNameInput'
      Cluster:
        Fn::ImportValue:
          !Sub "${StackName}-ECSCluster"  # output parameter from another
 CloudFormation template
      LaunchType: FARGATE
      DeploymentConfiguration:
```

```
        MaximumPercent: 200
        MinimumHealthyPercent: 75
      DesiredCount: !Ref 'TaskCountInput'
      NetworkConfiguration:
        AwsvpcConfiguration:
          AssignPublicIp: ENABLED
          SecurityGroups:
            - Fn::ImportValue:
                !Sub "${StackName}-ContainerSecurityGroup"    # output parameter from
another CloudFormation template
          Subnets:
            - Fn::ImportValue:r CloudFormation template
      TaskRoleArn: !Ref "AWS::NoValue"
      ContainerDefinitions:
        - Name: !Ref 'TaskNameInput'
          Cpu: !FindInMap [TaskSizeMap, !Ref 'TaskSizeInput', cpu]
          Memory: !FindInMap [TaskSizeMap, !Ref 'TaskSizeInput', memory]
          Image: !Ref 'ContainerImageInput'             # input parameter
          PortMappings:
            - ContainerPort: !Ref 'ContainerPortInput'  # input parameter

          LogConfiguration:
            LogDriver: 'awslogs'
            Options:
              awslogs-group: !Ref 'TaskNameInput'
                !Sub "${StackName}-PublicSubnetOne"    # output parameter from another
CloudFormation template
            - Fn::ImportValue:
                !Sub "${StackName}-PublicSubnetTwo"    # output parameter from another
CloudFormation template
      TaskDefinition: !Ref 'TaskDefinition'
      LoadBalancers:
        - ContainerName: !Ref 'TaskNameInput'
          ContainerPort: !Ref 'ContainerPortInput'  # input parameter
          TargetGroupArn: !Ref 'TargetGroup'

  # A target group. This is used for keeping track of all the tasks, and
  # what IP addresses / port numbers they have. You can query it yourself,
  # to use the addresses yourself, but most often this target group is just
  # connected to an application load balancer, or network load balancer, so
  # it can automatically distribute traffic across all the targets.
  TargetGroup:
    Type: AWS::ElasticLoadBalancingV2::TargetGroup
    Properties:
```

```
      HealthCheckIntervalSeconds: 6
      HealthCheckPath: /
      HealthCheckProtocol: HTTP
      HealthCheckTimeoutSeconds: 5
      HealthyThresholdCount: 2
      TargetType: ip
      Name: !Ref 'TaskNameInput'
      Port: !Ref 'ContainerPortInput'
      Protocol: HTTP
      UnhealthyThresholdCount: 2
      VpcId:
        Fn::ImportValue:
          !Sub "${StackName}-VPC"    # output parameter from another CloudFormation
template

  # Create a rule on the load balancer for routing traffic to the target group
  LoadBalancerRule:
    Type: AWS::ElasticLoadBalancingV2::ListenerRule
    Properties:
      Actions:
        - TargetGroupArn: !Ref 'TargetGroup'
          Type: 'forward'
      Conditions:
        - Field: path-pattern
          Values:
            - '*'
      ListenerArn: !Ref PublicLoadBalancerListener
      Priority: 1

  # Enable autoscaling for this service
  ScalableTarget:
    Type: AWS::ApplicationAutoScaling::ScalableTarget
    DependsOn: Service
    Properties:
      ServiceNamespace: 'ecs'
      ScalableDimension: 'ecs:service:DesiredCount'
      ResourceId:
        Fn::Join:
          - '/'
          - - service
            - Fn::ImportValue:
                !Sub "${StackName}-ECSCluster"
            - !Ref 'TaskNameInput'
      MinCapacity: 1
```

```
      MaxCapacity: 10
      RoleARN: !Sub arn:aws:iam::${AWS::AccountId}:role/
aws-service-role/ecs.application-autoscaling.amazonaws.com/
AWSServiceRoleForApplicationAutoScaling_ECSService

  # Create scaling policies for the service
  ScaleDownPolicy:
    Type: AWS::ApplicationAutoScaling::ScalingPolicy
    DependsOn: ScalableTarget
    Properties:
      PolicyName:
        Fn::Join:
          - '/'
          - - scale
            - !Ref 'TaskNameInput'
            - down
      PolicyType: StepScaling
      ResourceId:
        Fn::Join:
          - '/'
          - - service
            - Fn::ImportValue:
                !Sub "${StackName}-ECSCluster"  # output parameter from another
 CloudFormation template
            - !Ref 'TaskNameInput'
      ScalableDimension: 'ecs:service:DesiredCount'
      ServiceNamespace: 'ecs'
      StepScalingPolicyConfiguration:
        AdjustmentType: 'ChangeInCapacity'
        StepAdjustments:
          - MetricIntervalUpperBound: 0
            ScalingAdjustment: -1
        MetricAggregationType: 'Average'
        Cooldown: 60

  ScaleUpPolicy:
    Type: AWS::ApplicationAutoScaling::ScalingPolicy
    DependsOn: ScalableTarget
    Properties:
      PolicyName:
        Fn::Join:
          - '/'
          - - scale
            - !Ref 'TaskNameInput'
```

```
            - up
      PolicyType: StepScaling
      ResourceId:
        Fn::Join:
          - '/'
          - - service
            - Fn::ImportValue:
                !Sub "${StackName}-ECSCluster"
            - !Ref 'TaskNameInput'
      ScalableDimension: 'ecs:service:DesiredCount'
      ServiceNamespace: 'ecs'
      StepScalingPolicyConfiguration:
        AdjustmentType: 'ChangeInCapacity'
        StepAdjustments:
          - MetricIntervalLowerBound: 0
            MetricIntervalUpperBound: 15
            ScalingAdjustment: 1
          - MetricIntervalLowerBound: 15
            MetricIntervalUpperBound: 25
            ScalingAdjustment: 2
          - MetricIntervalLowerBound: 25
            ScalingAdjustment: 3
        MetricAggregationType: 'Average'
        Cooldown: 60

  # Create alarms to trigger these policies
  LowCpuUsageAlarm:
    Type: AWS::CloudWatch::Alarm
    Properties:
      AlarmName:
        Fn::Join:
          - '-'
          - - low-cpu
            - !Ref 'TaskNameInput'
      AlarmDescription:
        Fn::Join:
          - ' '
          - - "Low CPU utilization for service"
            - !Ref 'TaskNameInput'
      MetricName: CPUUtilization
      Namespace: AWS/ECS
      Dimensions:
        - Name: ServiceName
          Value: !Ref 'TaskNameInput'
```

```
        - Name: ClusterName
          Value:
            Fn::ImportValue:
              !Sub "${StackName}-ECSCluster"
      Statistic: Average
      Period: 60
      EvaluationPeriods: 1
      Threshold: 20
      ComparisonOperator: LessThanOrEqualToThreshold
      AlarmActions:
        - !Ref ScaleDownPolicy

  HighCpuUsageAlarm:
    Type: AWS::CloudWatch::Alarm
    Properties:
      AlarmName:
        Fn::Join:
          - '-'
          - - high-cpu
            - !Ref 'TaskNameInput'
      AlarmDescription:
        Fn::Join:
          - ' '
          - - "High CPU utilization for service"
            - !Ref 'TaskNameInput'
      MetricName: CPUUtilization
      Namespace: AWS/ECS
      Dimensions:
        - Name: ServiceName
          Value: !Ref 'TaskNameInput'
        - Name: ClusterName
          Value:
            Fn::ImportValue:
              !Sub "${StackName}-ECSCluster"
      Statistic: Average
      Period: 60
      EvaluationPeriods: 1
      Threshold: 70
      ComparisonOperator: GreaterThanOrEqualToThreshold
      AlarmActions:
        - !Ref ScaleUpPolicy

  EcsSecurityGroupIngressFromPublicALB:
    Type: AWS::EC2::SecurityGroupIngress
```

```
    Properties:
      Description: Ingress from the public ALB
      GroupId:
        Fn::ImportValue:
          !Sub "${StackName}-ContainerSecurityGroup"
      IpProtocol: -1
      SourceSecurityGroupId: !Ref 'PublicLoadBalancerSG'

  # Public load balancer, hosted in public subnets that is accessible
  # to the public, and is intended to route traffic to one or more public
  # facing services. This is used for accepting traffic from the public
  # internet and directing it to public facing microservices
  PublicLoadBalancerSG:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Access to the public facing load balancer
      VpcId:
        Fn::ImportValue:
          !Sub "${StackName}-VPC"
      SecurityGroupIngress:
          # Allow access to ALB from anywhere on the internet
          - CidrIp: 0.0.0.0/0
            IpProtocol: -1

  PublicLoadBalancer:
    Type: AWS::ElasticLoadBalancingV2::LoadBalancer
    Properties:
      Scheme: internet-facing
      LoadBalancerAttributes:
      - Key: idle_timeout.timeout_seconds
        Value: '30'
      Subnets:
        # The load balancer is placed into the public subnets, so that traffic
        # from the internet can reach the load balancer directly via the internet
gateway
        - Fn::ImportValue:
            !Sub "${StackName}-PublicSubnetOne"
        - Fn::ImportValue:
            !Sub "${StackName}-PublicSubnetTwo"
      SecurityGroups: [!Ref 'PublicLoadBalancerSG']

  PublicLoadBalancerListener:
    Type: AWS::ElasticLoadBalancingV2::Listener
    DependsOn:
```

```
      - PublicLoadBalancer
    Properties:
      DefaultActions:
        - TargetGroupArn: !Ref 'TargetGroup'
          Type: 'forward'
      LoadBalancerArn: !Ref 'PublicLoadBalancer'
      Port: 80
      Protocol: HTTP
# These output values will be available to other templates to use.
Outputs:
  ServiceEndpoint:        # output
    Description: The URL to access the service
    Value: !Sub "http://${PublicLoadBalancer.DNSName}"
```

これらのファイルは、 で使用するように調整できます AWS Proton。

インフラストラクチャをコードとして に持ち込む AWS Proton

わずかな変更を加えると、 が環境のデプロイ AWS Proton に使用する環境テンプレートバンドルの Infrastructure as Code (IaC) ファイルとして例 1 を使用できます (例 3 を参照) 。

CloudFormation パラメータを使用する代わりに、Jinja 構文を使用して、Open API ベースのスキーマファイルに定義したパラメータを参照します。これらの入力パラメータは使いやすいようコメント化されており、IaC ファイル内で複数回参照されます。これにより、 はパラメータ値を監査およびチェック AWS Proton できます。また、1 つの IaC ファイルの出力パラメータ値を別の IaC ファイルのパラメータと照合して一致すれば挿入することもできます。

管理者は、名前空間を入力 AWS Proton `environment.inputs.`パラメータに追加できます。サービス IaC ファイル内で環境 IaC ファイル出力を参照するには、`environment.outputs.` 名前空間を出力に追加します (たとえば、`environment.outputs.ClusterName`)。最後に、中括弧と引用符で囲みます。

これらの変更により、CloudFormation IaC ファイルを で使用できます AWS Proton。

例 3: AWS Proton コードファイルとしての環境インフラストラクチャ

```
AWSTemplateFormatVersion: '2010-09-09'
Description: AWS Fargate cluster running containers in a public subnet. Only supports
             public facing load balancer, and public service discovery prefixes.
Mappings:
  # The VPC and subnet configuration is passed in via the environment spec.
  SubnetConfig:
    VPC:
```

```
      CIDR: '{{ environment.inputs.vpc_cidr}}'        # input parameter
    PublicOne:
      CIDR: '{{ environment.inputs.subnet_one_cidr}}' # input parameter
    PublicTwo:
      CIDR: '{{ environment.inputs.subnet_two_cidr}}' # input parameter
Resources:
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      EnableDnsSupport: true
      EnableDnsHostnames: true
      CidrBlock: !FindInMap ['SubnetConfig', 'VPC', 'CIDR']

  # Two public subnets, where containers will have public IP addresses
  PublicSubnetOne:
    Type: AWS::EC2::Subnet
    Properties:
      AvailabilityZone:
         Fn::Select:
         - 0
         - Fn::GetAZs: {Ref: 'AWS::Region'}
      VpcId: !Ref 'VPC'
      CidrBlock: !FindInMap ['SubnetConfig', 'PublicOne', 'CIDR']
      MapPublicIpOnLaunch: true

  PublicSubnetTwo:
    Type: AWS::EC2::Subnet
    Properties:
      AvailabilityZone:
         Fn::Select:
         - 1
         - Fn::GetAZs: {Ref: 'AWS::Region'}
      VpcId: !Ref 'VPC'
      CidrBlock: !FindInMap ['SubnetConfig', 'PublicTwo', 'CIDR']
      MapPublicIpOnLaunch: true

  # Setup networking resources for the public subnets. Containers
  # in the public subnets have public IP addresses and the routing table
  # sends network traffic via the internet gateway.
  InternetGateway:
    Type: AWS::EC2::InternetGateway
  GatewayAttachement:
    Type: AWS::EC2::VPCGatewayAttachment
    Properties:
```

```
      VpcId: !Ref 'VPC'
      InternetGatewayId: !Ref 'InternetGateway'
  PublicRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref 'VPC'
  PublicRoute:
    Type: AWS::EC2::Route
    DependsOn: GatewayAttachement
    Properties:
      RouteTableId: !Ref 'PublicRouteTable'
      DestinationCidrBlock: '0.0.0.0/0'
      GatewayId: !Ref 'InternetGateway'
  PublicSubnetOneRouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      SubnetId: !Ref PublicSubnetOne
      RouteTableId: !Ref PublicRouteTable
  PublicSubnetTwoRouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      SubnetId: !Ref PublicSubnetTwo
      RouteTableId: !Ref PublicRouteTable

  # ECS Resources
  ECSCluster:
    Type: AWS::ECS::Cluster

  # A security group for the containers we will run in Fargate.
  # Rules are added to this security group based on what ingress you
  # add for the cluster.
  ContainerSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Access to the Fargate containers
      VpcId: !Ref 'VPC'

  # This is a role which is used by the ECS tasks themselves.
  ECSTaskExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
        - Effect: Allow
```

```
          Principal:
            Service: [ecs-tasks.amazonaws.com]
          Action: ['sts:AssumeRole']
      Path: /
      ManagedPolicyArns:
        - 'arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy'

# These output values are available to service infrastructure as code files as outputs,
 when given the
# the 'service_instance.environment.outputs.' namespace, for example,
 service_instance.environment.outputs.ClusterName.

Outputs:
  ClusterName:                                          # output
    Description: The name of the ECS cluster
    Value: !Ref 'ECSCluster'
  ECSTaskExecutionRole:                                 # output
    Description: The ARN of the ECS role
    Value: !GetAtt 'ECSTaskExecutionRole.Arn'
  VpcId:                                                # output
    Description: The ID of the VPC that this stack is deployed in
    Value: !Ref 'VPC'
  PublicSubnetOne:                                      # output
    Description: Public subnet one
    Value: !Ref 'PublicSubnetOne'
  PublicSubnetTwo:                                      # output
    Description: Public subnet two
    Value: !Ref 'PublicSubnetTwo'
  ContainerSecurityGroup:                               # output
    Description: A security group used to allow Fargate containers to receive traffic
    Value: !Ref 'ContainerSecurityGroup'
```

例 1 と例 3 の IaC ファイルから生成される CloudFormation スタックは少し異なります。スタックテンプレートファイルでは、パラメータの表示が異なります。例 1 CloudFormation スタックテンプレートファイルは、スタックテンプレートビューにパラメータラベル (キー) を表示します。サンプル 3 AWS Proton CloudFormation インフラストラクチャスタックテンプレートファイルにはパラメータ値が表示されます。 AWS Proton 入力パラメータはコンソールの CloudFormation スタックパラメータビューに表示されません。 CloudFormation

例 4 では、 AWS Proton サービス IaC ファイルは例 2 に対応しています。

例 4: AWS Proton サービスインスタンス IaC ファイル

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Deploy a service on AWS Fargate, hosted in a public subnet, and accessible
 via a public load balancer.
Mappings:
  TaskSize:
    x-small:
      cpu: 256
      memory: 512
    small:
      cpu: 512
      memory: 1024
    medium:
      cpu: 1024
      memory: 2048
    large:
      cpu: 2048
      memory: 4096
    x-large:
      cpu: 4096
      memory: 8192
Resources:
  # A log group for storing the stdout logs from this service's containers
  LogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: '{{service_instance.name}}' # resource parameter

  # The task definition. This is a simple metadata description of what
  # container to run, and what resource requirements it has.
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      Family: '{{service_instance.name}}'
      Cpu: !FindInMap [TaskSize, {{service_instance.inputs.task_size}}, cpu] # input
 parameter
      Memory: !FindInMap [TaskSize, {{service_instance.inputs.task_size}}, memory]
      NetworkMode: awsvpc
      RequiresCompatibilities:
        - FARGATE
      ExecutionRoleArn: '{{environment.outputs.ECSTaskExecutionRole}}' # output from an
 environment infrastructure as code file
      TaskRoleArn: !Ref "AWS::NoValue"
```

```
      ContainerDefinitions:
        - Name: '{{service_instance.name}}'
          Cpu: !FindInMap [TaskSize, {{service_instance.inputs.task_size}}, cpu]
          Memory: !FindInMap [TaskSize, {{service_instance.inputs.task_size}}, memory]
          Image: '{{service_instance.inputs.image}}'
          PortMappings:
            - ContainerPort: '{{service_instance.inputs.port}}' # input parameter
          LogConfiguration:
            LogDriver: 'awslogs'
            Options:
              awslogs-group: '{{service_instance.name}}'
              awslogs-region: !Ref 'AWS::Region'
              awslogs-stream-prefix: '{{service_instance.name}}'

  # The service_instance. The service is a resource which allows you to run multiple
  # copies of a type of task, and gather up their logs and metrics, as well
  # as monitor the number of running tasks and replace any that have crashed
  Service:
    Type: AWS::ECS::Service
    DependsOn: LoadBalancerRule
    Properties:
      ServiceName: '{{service_instance.name}}'
      Cluster: '{{environment.outputs.ClusterName}}' # output from an environment
infrastructure as code file
      LaunchType: FARGATE
      DeploymentConfiguration:
        MaximumPercent: 200
        MinimumHealthyPercent: 75
      DesiredCount: '{{service_instance.inputs.desired_count}}'       # input parameter
      NetworkConfiguration:
        AwsvpcConfiguration:
          AssignPublicIp: ENABLED
          SecurityGroups:
            - '{{environment.outputs.ContainerSecurityGroup}}' # output from an
environment infrastructure as code file
          Subnets:
            - '{{environment.outputs.PublicSubnetOne}}'        # output from an
environment infrastructure as code file
            - '{{environment.outputs.PublicSubnetTwo}}'
      TaskDefinition: !Ref 'TaskDefinition'
      LoadBalancers:
        - ContainerName: '{{service_instance.name}}'
          ContainerPort: '{{service_instance.inputs.port}}'
          TargetGroupArn: !Ref 'TargetGroup'
```

```
  # A target group. This is used for keeping track of all the tasks, and
  # what IP addresses / port numbers they have. You can query it yourself,
  # to use the addresses yourself, but most often this target group is just
  # connected to an application load balancer, or network load balancer, so
  # it can automatically distribute traffic across all the targets.
  TargetGroup:
    Type: AWS::ElasticLoadBalancingV2::TargetGroup
    Properties:
      HealthCheckIntervalSeconds: 6
      HealthCheckPath: /
      HealthCheckProtocol: HTTP
      HealthCheckTimeoutSeconds: 5
      HealthyThresholdCount: 2
      TargetType: ip
      Name: '{{service_instance.name}}'
      Port: '{{service_instance.inputs.port}}'
      Protocol: HTTP
      UnhealthyThresholdCount: 2
      VpcId: '{{environment.outputs.VpcId}}' # output from an environment
infrastructure as code file

  # Create a rule on the load balancer for routing traffic to the target group
  LoadBalancerRule:
    Type: AWS::ElasticLoadBalancingV2::ListenerRule
    Properties:
      Actions:
        - TargetGroupArn: !Ref 'TargetGroup'
          Type: 'forward'
      Conditions:
        - Field: path-pattern
          Values:
            - '*'
      ListenerArn: !Ref PublicLoadBalancerListener
      Priority: 1

  # Enable autoscaling for this service
  ScalableTarget:
    Type: AWS::ApplicationAutoScaling::ScalableTarget
    DependsOn: Service
    Properties:
      ServiceNamespace: 'ecs'
      ScalableDimension: 'ecs:service:DesiredCount'
      ResourceId:
```

```
        Fn::Join:
          - '/'
          - - service
            - '{{environment.outputs.ClusterName}}' # output from an environment
 infrastructure as code file
            - '{{service_instance.name}}'
      MinCapacity: 1
      MaxCapacity: 10
      RoleARN: !Sub arn:aws:iam::${AWS::AccountId}:role/
aws-service-role/ecs.application-autoscaling.amazonaws.com/
AWSServiceRoleForApplicationAutoScaling_ECSService

  # Create scaling policies for the service
  ScaleDownPolicy:
    Type: AWS::ApplicationAutoScaling::ScalingPolicy
    DependsOn: ScalableTarget
    Properties:
      PolicyName:
        Fn::Join:
          - '/'
          - - scale
            - '{{service_instance.name}}'
            - down
      PolicyType: StepScaling
      ResourceId:
        Fn::Join:
          - '/'
          - - service
            - '{{environment.outputs.ClusterName}}'
            - '{{service_instance.name}}'
      ScalableDimension: 'ecs:service:DesiredCount'
      ServiceNamespace: 'ecs'
      StepScalingPolicyConfiguration:
        AdjustmentType: 'ChangeInCapacity'
        StepAdjustments:
          - MetricIntervalUpperBound: 0
            ScalingAdjustment: -1
        MetricAggregationType: 'Average'
        Cooldown: 60

  ScaleUpPolicy:
    Type: AWS::ApplicationAutoScaling::ScalingPolicy
    DependsOn: ScalableTarget
    Properties:
```

```
      PolicyName:
        Fn::Join:
          - '/'
          - - scale
            - '{{service_instance.name}}'
            - up
      PolicyType: StepScaling
      ResourceId:
        Fn::Join:
          - '/'
          - - service
            - '{{environment.outputs.ClusterName}}'
            - '{{service_instance.name}}'
      ScalableDimension: 'ecs:service:DesiredCount'
      ServiceNamespace: 'ecs'
      StepScalingPolicyConfiguration:
        AdjustmentType: 'ChangeInCapacity'
        StepAdjustments:
          - MetricIntervalLowerBound: 0
            MetricIntervalUpperBound: 15
            ScalingAdjustment: 1
          - MetricIntervalLowerBound: 15
            MetricIntervalUpperBound: 25
            ScalingAdjustment: 2
          - MetricIntervalLowerBound: 25
            ScalingAdjustment: 3
        MetricAggregationType: 'Average'
        Cooldown: 60

  # Create alarms to trigger these policies
  LowCpuUsageAlarm:
    Type: AWS::CloudWatch::Alarm
    Properties:
      AlarmName:
        Fn::Join:
          - '-'
          - - low-cpu
            - '{{service_instance.name}}'
      AlarmDescription:
        Fn::Join:
          - ' '
          - - "Low CPU utilization for service"
            - '{{service_instance.name}}'
      MetricName: CPUUtilization
```

```
      Namespace: AWS/ECS
      Dimensions:
        - Name: ServiceName
          Value: '{{service_instance.name}}'
        - Name: ClusterName
          Value:
            '{{environment.outputs.ClusterName}}'
      Statistic: Average
      Period: 60
      EvaluationPeriods: 1
      Threshold: 20
      ComparisonOperator: LessThanOrEqualToThreshold
      AlarmActions:
        - !Ref ScaleDownPolicy

  HighCpuUsageAlarm:
    Type: AWS::CloudWatch::Alarm
    Properties:
      AlarmName:
        Fn::Join:
          - '-'
          - - high-cpu
            - '{{service_instance.name}}'
      AlarmDescription:
        Fn::Join:
          - ' '
          - - "High CPU utilization for service"
            - '{{service_instance.name}}'
      MetricName: CPUUtilization
      Namespace: AWS/ECS
      Dimensions:
        - Name: ServiceName
          Value: '{{service_instance.name}}'
        - Name: ClusterName
          Value:
            '{{environment.outputs.ClusterName}}'
      Statistic: Average
      Period: 60
      EvaluationPeriods: 1
      Threshold: 70
      ComparisonOperator: GreaterThanOrEqualToThreshold
      AlarmActions:
        - !Ref ScaleUpPolicy
```

```
  EcsSecurityGroupIngressFromPublicALB:
    Type: AWS::EC2::SecurityGroupIngress
    Properties:
      Description: Ingress from the public ALB
      GroupId: '{{environment.outputs.ContainerSecurityGroup}}'
      IpProtocol: -1
      SourceSecurityGroupId: !Ref 'PublicLoadBalancerSG'

  # Public load balancer, hosted in public subnets that is accessible
  # to the public, and is intended to route traffic to one or more public
  # facing services. This is used for accepting traffic from the public
  # internet and directing it to public facing microservices
  PublicLoadBalancerSG:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Access to the public facing load balancer
      VpcId: '{{environment.outputs.VpcId}}'
      SecurityGroupIngress:
          # Allow access to ALB from anywhere on the internet
          - CidrIp: 0.0.0.0/0
            IpProtocol: -1

  PublicLoadBalancer:
    Type: AWS::ElasticLoadBalancingV2::LoadBalancer
    Properties:
      Scheme: internet-facing
      LoadBalancerAttributes:
      - Key: idle_timeout.timeout_seconds
        Value: '30'
      Subnets:
        # The load balancer is placed into the public subnets, so that traffic
        # from the internet can reach the load balancer directly via the internet
gateway
        - '{{environment.outputs.PublicSubnetOne}}'
        - '{{environment.outputs.PublicSubnetTwo}}'
      SecurityGroups: [!Ref 'PublicLoadBalancerSG']

  PublicLoadBalancerListener:
    Type: AWS::ElasticLoadBalancingV2::Listener
    DependsOn:
      - PublicLoadBalancer
    Properties:
      DefaultActions:
        - TargetGroupArn: !Ref 'TargetGroup'
```

```
          Type: 'forward'
      LoadBalancerArn: !Ref 'PublicLoadBalancer'
      Port: 80
      Protocol: HTTP
Outputs:
  ServiceEndpoint:         # output
    Description: The URL to access the service
    Value: !Sub "http://${PublicLoadBalancer.DNSName}"
```

例 5 では、 AWS Proton パイプライン IaC ファイルは、例 4 でプロビジョニングされたサービスインスタンスをサポートするようにパイプラインインフラストラクチャをプロビジョニングします。

例 5: AWS Proton サービスパイプライン IaC ファイル

```
Resources:
  ECRRepo:
    Type: AWS::ECR::Repository
    DeletionPolicy: Retain
  BuildProject:
    Type: AWS::CodeBuild::Project
    Properties:
      Artifacts:
        Type: CODEPIPELINE
      Environment:
        ComputeType: BUILD_GENERAL1_SMALL
        Image: aws/codebuild/amazonlinux2-x86_64-standard:3.0
        PrivilegedMode: true
        Type: LINUX_CONTAINER
        EnvironmentVariables:
        - Name: repo_name
          Type: PLAINTEXT
          Value: !Ref ECRRepo
        - Name: service_name
          Type: PLAINTEXT
          Value: '{{ service.name }}'    # resource parameter
      ServiceRole:
        Fn::GetAtt:
          - PublishRole
          - Arn
      Source:
        BuildSpec:
          Fn::Join:
            - ""
```

```
            - - >-
                {
                  "version": "0.2",
                  "phases": {
                    "install": {
                      "runtime-versions": {
                        "docker": 18
                      },
                      "commands": [
                        "pip3 install --upgrade --user awscli",
                        "echo
 'f6bd1536a743ab170b35c94ed4c7c4479763356bd543af5d391122f4af852460  yq_linux_amd64' >
 yq_linux_amd64.sha",
                        "wget https://github.com/mikefarah/yq/releases/download/3.4.0/
yq_linux_amd64",
                        "sha256sum -c yq_linux_amd64.sha",
                        "mv yq_linux_amd64 /usr/bin/yq",
                        "chmod +x /usr/bin/yq"
                      ]
                    },
                    "pre_build": {
                      "commands": [
                        "cd $CODEBUILD_SRC_DIR",
                        "$(aws ecr get-login --no-include-email --region
 $AWS_DEFAULT_REGION)",
                        "{{ pipeline.inputs.unit_test_command }}",    # input parameter
                      ]
                    },
                    "build": {
                      "commands": [
                        "IMAGE_REPO_NAME=$repo_name",
                        "IMAGE_TAG=$CODEBUILD_BUILD_NUMBER",
                        "IMAGE_ID=
              - Ref: AWS::AccountId
              - >-
                .dkr.ecr.$AWS_DEFAULT_REGION.amazonaws.com/$IMAGE_REPO_NAME:
$IMAGE_TAG",
                        "docker build -t $IMAGE_REPO_NAME:$IMAGE_TAG -f
 {{ pipeline.inputs.dockerfile }} .",     # input parameter
                        "docker tag $IMAGE_REPO_NAME:$IMAGE_TAG $IMAGE_ID;",
                        "docker push $IMAGE_ID"
                      ]
                    },
                    "post_build": {
```

```
                    "commands": [
                      "aws proton --region $AWS_DEFAULT_REGION get-service --name
 $service_name | jq -r .service.spec > service.yaml",
                      "yq w service.yaml 'instances[*].spec.image' \"$IMAGE_ID\" >
 rendered_service.yaml"
                    ]
                  }
                },
                "artifacts": {
                  "files": [
                    "rendered_service.yaml"
                  ]
                }
              }
        Type: CODEPIPELINE
      EncryptionKey:
        Fn::GetAtt:
          - PipelineArtifactsBucketEncryptionKey
          - Arn
{% for service_instance in service_instances %}
  Deploy{{loop.index}}Project:
    Type: AWS::CodeBuild::Project
    Properties:
      Artifacts:
        Type: CODEPIPELINE
      Environment:
        ComputeType: BUILD_GENERAL1_SMALL
        Image: aws/codebuild/amazonlinux2-x86_64-standard:3.0
        PrivilegedMode: false
        Type: LINUX_CONTAINER
        EnvironmentVariables:
        - Name: service_name
          Type: PLAINTEXT
          Value:  '{{service.name}}'          # resource parameter
        - Name: service_instance_name
          Type: PLAINTEXT
          Value: '{{service_instance.name}}'  # resource parameter
      ServiceRole:
        Fn::GetAtt:
          - DeploymentRole
          - Arn
      Source:
        BuildSpec: >-
          {
```

```
            "version": "0.2",
            "phases": {
              "build": {
                "commands": [
                  "pip3 install --upgrade --user awscli",
                  "aws proton --region $AWS_DEFAULT_REGION update-service-instance
 --deployment-type CURRENT_VERSION --name $service_instance_name --service-name
 $service_name --spec file://rendered_service.yaml",
                  "aws proton --region $AWS_DEFAULT_REGION wait service-instance-
deployed --name $service_instance_name --service-name $service_name"
                ]
              }
            }
          }
        Type: CODEPIPELINE
      EncryptionKey:
        Fn::GetAtt:
          - PipelineArtifactsBucketEncryptionKey
          - Arn
{% endfor %}
  # This role is used to build and publish an image to ECR
  PublishRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              Service: codebuild.amazonaws.com
        Version: "2012-10-17"
  PublishRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - logs:CreateLogGroup
              - logs:CreateLogStream
              - logs:PutLogEvents
            Effect: Allow
            Resource:
              - Fn::Join:
                  - ""
```

```
                - - "arn:"
                  - Ref: AWS::Partition
                  - ":logs:"
                  - Ref: AWS::Region
                  - ":"
                  - Ref: AWS::AccountId
                  - :log-group:/aws/codebuild/
                  - Ref: BuildProject
              - Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":logs:"
                    - Ref: AWS::Region
                    - ":"
                    - Ref: AWS::AccountId
                    - :log-group:/aws/codebuild/
                    - Ref: BuildProject
                    - :*
          - Action:
              - codebuild:CreateReportGroup
              - codebuild:CreateReport
              - codebuild:UpdateReport
              - codebuild:BatchPutTestCases
            Effect: Allow
            Resource:
              Fn::Join:
                - ""
                - - "arn:"
                  - Ref: AWS::Partition
                  - ":codebuild:"
                  - Ref: AWS::Region
                  - ":"
                  - Ref: AWS::AccountId
                  - :report-group/
                  - Ref: BuildProject
                  - -*
          - Action:
              - ecr:GetAuthorizationToken
            Effect: Allow
            Resource: "*"
          - Action:
              - ecr:BatchCheckLayerAvailability
              - ecr:CompleteLayerUpload
```

```
            - ecr:GetAuthorizationToken
            - ecr:InitiateLayerUpload
            - ecr:PutImage
            - ecr:UploadLayerPart
          Effect: Allow
          Resource:
            Fn::GetAtt:
              - ECRRepo
              - Arn
        - Action:
            - proton:GetService
          Effect: Allow
          Resource: "*"
        - Action:
            - s3:GetObject*
            - s3:GetBucket*
            - s3:List*
            - s3:DeleteObject*
            - s3:PutObject*
            - s3:Abort*
          Effect: Allow
          Resource:
            - Fn::GetAtt:
                - PipelineArtifactsBucket
                - Arn
            - Fn::Join:
                - ""
                - - Fn::GetAtt:
                      - PipelineArtifactsBucket
                      - Arn
                  - /*
        - Action:
            - kms:Decrypt
            - kms:DescribeKey
            - kms:Encrypt
            - kms:ReEncrypt*
            - kms:GenerateDataKey*
          Effect: Allow
          Resource:
            Fn::GetAtt:
              - PipelineArtifactsBucketEncryptionKey
              - Arn
        - Action:
            - kms:Decrypt
```

```
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
        Version: "2012-10-17"
      PolicyName: PublishRoleDefaultPolicy
      Roles:
        - Ref: PublishRole

  DeploymentRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              Service: codebuild.amazonaws.com
        Version: "2012-10-17"
  DeploymentRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - logs:CreateLogGroup
              - logs:CreateLogStream
              - logs:PutLogEvents
            Effect: Allow
            Resource:
              - Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":logs:"
                    - Ref: AWS::Region
                    - ":"
                    - Ref: AWS::AccountId
                    - :log-group:/aws/codebuild/Deploy*Project*
              - Fn::Join:
```

```
                - ""
                - - "arn:"
                  - Ref: AWS::Partition
                  - ":logs:"
                  - Ref: AWS::Region
                  - ":"
                  - Ref: AWS::AccountId
                  - :log-group:/aws/codebuild/Deploy*Project:*
            - Action:
                - codebuild:CreateReportGroup
                - codebuild:CreateReport
                - codebuild:UpdateReport
                - codebuild:BatchPutTestCases
              Effect: Allow
              Resource:
                Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":codebuild:"
                    - Ref: AWS::Region
                    - ":"
                    - Ref: AWS::AccountId
                    - :report-group/Deploy*Project
                    - -*
            - Action:
                - proton:UpdateServiceInstance
                - proton:GetServiceInstance
              Effect: Allow
              Resource: "*"
            - Action:
                - s3:GetObject*
                - s3:GetBucket*
                - s3:List*
              Effect: Allow
              Resource:
                - Fn::GetAtt:
                    - PipelineArtifactsBucket
                    - Arn
                - Fn::Join:
                    - ""
                    - - Fn::GetAtt:
                          - PipelineArtifactsBucket
                          - Arn
```

```
                    - /*
          - Action:
              - kms:Decrypt
              - kms:DescribeKey
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
          - Action:
              - kms:Decrypt
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
        Version: "2012-10-17"
      PolicyName: DeploymentRoleDefaultPolicy
      Roles:
        - Ref: DeploymentRole
  PipelineArtifactsBucketEncryptionKey:
    Type: AWS::KMS::Key
    Properties:
      KeyPolicy:
        Statement:
          - Action:
              - kms:Create*
              - kms:Describe*
              - kms:Enable*
              - kms:List*
              - kms:Put*
              - kms:Update*
              - kms:Revoke*
              - kms:Disable*
              - kms:Get*
              - kms:Delete*
              - kms:ScheduleKeyDeletion
              - kms:CancelKeyDeletion
              - kms:GenerateDataKey
              - kms:TagResource
              - kms:UntagResource
```

```
            Effect: Allow
            Principal:
              AWS:
                Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":iam::"
                    - Ref: AWS::AccountId
                    - :root
            Resource: "*"
          - Action:
              - kms:Decrypt
              - kms:DescribeKey
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Principal:
              AWS:
                Fn::GetAtt:
                  - PipelineRole
                  - Arn
            Resource: "*"
          - Action:
              - kms:Decrypt
              - kms:DescribeKey
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Principal:
              AWS:
                Fn::GetAtt:
                  - PublishRole
                  - Arn
            Resource: "*"
          - Action:
              - kms:Decrypt
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Principal:
```

```
            AWS:
              Fn::GetAtt:
                - PublishRole
                - Arn
          Resource: "*"
        - Action:
            - kms:Decrypt
            - kms:DescribeKey
          Effect: Allow
          Principal:
            AWS:
              Fn::GetAtt:
                - DeploymentRole
                - Arn
          Resource: "*"
        - Action:
            - kms:Decrypt
            - kms:Encrypt
            - kms:ReEncrypt*
            - kms:GenerateDataKey*
          Effect: Allow
          Principal:
            AWS:
              Fn::GetAtt:
                - DeploymentRole
                - Arn
          Resource: "*"
      Version: "2012-10-17"
  UpdateReplacePolicy: Delete
  DeletionPolicy: Delete
PipelineArtifactsBucket:
  Type: AWS::S3::Bucket
  Properties:
    VersioningConfiguration:
      Status: Enabled
    BucketEncryption:
      ServerSideEncryptionConfiguration:
        - ServerSideEncryptionByDefault:
            KMSMasterKeyID:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
            SSEAlgorithm: aws:kms
    PublicAccessBlockConfiguration:
```

```
        BlockPublicAcls: true
        BlockPublicPolicy: true
        IgnorePublicAcls: true
        RestrictPublicBuckets: true
    UpdateReplacePolicy: Retain
    DeletionPolicy: Retain
  PipelineArtifactsBucketEncryptionKeyAlias:
    Type: AWS::KMS::Alias
    Properties:
      AliasName: 'alias/codepipeline-encryption-key-{{ service.name }}'
      TargetKeyId:
        Fn::GetAtt:
          - PipelineArtifactsBucketEncryptionKey
          - Arn
    UpdateReplacePolicy: Delete
    DeletionPolicy: Delete
  PipelineRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              Service: codepipeline.amazonaws.com
        Version: "2012-10-17"
  PipelineRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - s3:GetObject*
              - s3:GetBucket*
              - s3:List*
              - s3:DeleteObject*
              - s3:PutObject*
              - s3:Abort*
            Effect: Allow
            Resource:
              - Fn::GetAtt:
                  - PipelineArtifactsBucket
                  - Arn
              - Fn::Join:
```

```
                - ""
                - - Fn::GetAtt:
                      - PipelineArtifactsBucket
                      - Arn
                  - /*
          - Action:
              - kms:Decrypt
              - kms:DescribeKey
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
          - Action: codestar-connections:*
            Effect: Allow
            Resource: "*"
          - Action: sts:AssumeRole
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineBuildCodePipelineActionRole
                - Arn
          - Action: sts:AssumeRole
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineDeployCodePipelineActionRole
                - Arn
        Version: "2012-10-17"
      PolicyName: PipelineRoleDefaultPolicy
      Roles:
        - Ref: PipelineRole
  Pipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      RoleArn:
        Fn::GetAtt:
          - PipelineRole
          - Arn
      Stages:
        - Actions:
```

```
            - ActionTypeId:
                Category: Source
                Owner: AWS
                Provider: CodeStarSourceConnection
                Version: "1"
              Configuration:
                ConnectionArn: '{{ service.repository_connection_arn }}'
                FullRepositoryId: '{{ service.repository_id }}'
                BranchName: '{{ service.branch_name }}'
              Name: Checkout
              OutputArtifacts:
                - Name: Artifact_Source_Checkout
              RunOrder: 1
          Name: Source
        - Actions:
            - ActionTypeId:
                Category: Build
                Owner: AWS
                Provider: CodeBuild
                Version: "1"
              Configuration:
                ProjectName:
                  Ref: BuildProject
              InputArtifacts:
                - Name: Artifact_Source_Checkout
              Name: Build
              OutputArtifacts:
                - Name: BuildOutput
              RoleArn:
                Fn::GetAtt:
                  - PipelineBuildCodePipelineActionRole
                  - Arn
              RunOrder: 1
          Name: Build {%- for service_instance in service_instances %}
        - Actions:
            - ActionTypeId:
                Category: Build
                Owner: AWS
                Provider: CodeBuild
                Version: "1"
              Configuration:
                ProjectName:
                  Ref: Deploy{{loop.index}}Project
              InputArtifacts:
```

```
                - Name: BuildOutput
              Name: Deploy
              RoleArn:
                Fn::GetAtt:
                  - PipelineDeployCodePipelineActionRole
                  - Arn
              RunOrder: 1
          Name: 'Deploy{{service_instance.name}}'
{%- endfor %}
      ArtifactStore:
        EncryptionKey:
          Id:
            Fn::GetAtt:
              - PipelineArtifactsBucketEncryptionKey
              - Arn
          Type: KMS
        Location:
          Ref: PipelineArtifactsBucket
        Type: S3
    DependsOn:
      - PipelineRoleDefaultPolicy
      - PipelineRole
  PipelineBuildCodePipelineActionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              AWS:
                Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":iam::"
                    - Ref: AWS::AccountId
                    - :root
        Version: "2012-10-17"
  PipelineBuildCodePipelineActionRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
```

```
          - Action:
              - codebuild:BatchGetBuilds
              - codebuild:StartBuild
              - codebuild:StopBuild
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - BuildProject
                - Arn
        Version: "2012-10-17"
      PolicyName: PipelineBuildCodePipelineActionRoleDefaultPolicy
      Roles:
        - Ref: PipelineBuildCodePipelineActionRole
  PipelineDeployCodePipelineActionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              AWS:
                Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":iam::"
                    - Ref: AWS::AccountId
                    - :root
        Version: "2012-10-17"
  PipelineDeployCodePipelineActionRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - codebuild:BatchGetBuilds
              - codebuild:StartBuild
              - codebuild:StopBuild
            Effect: Allow
            Resource:
              Fn::Join:
                - ""
                - - "arn:"
```

```
                  - Ref: AWS::Partition
                  - ":codebuild:"
                  - Ref: AWS::Region
                  - ":"
                  - Ref: AWS::AccountId
                  - ":project/Deploy*"
        Version: "2012-10-17"
      PolicyName: PipelineDeployCodePipelineActionRoleDefaultPolicy
      Roles:
        - Ref: PipelineDeployCodePipelineActionRole
Outputs:
  PipelineEndpoint:
    Description: The URL to access the pipeline
    Value: !Sub "https://${AWS::Region}.console.aws.amazon.com/codesuite/codepipeline/
pipelines/${Pipeline}/view?region=${AWS::Region}"

                ]
              }
            }
          }
        Type: CODEPIPELINE
      EncryptionKey:
        Fn::GetAtt:
          - PipelineArtifactsBucketEncryptionKey
          - Arn
{% endfor %}
  # This role is used to build and publish an image to ECR
  PublishRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              Service: codebuild.amazonaws.com
        Version: "2012-10-17"
  PublishRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - logs:CreateLogGroup
```

```
            - logs:CreateLogStream
            - logs:PutLogEvents
          Effect: Allow
          Resource:
            - Fn::Join:
                - ""
                - - "arn:"
                  - Ref: AWS::Partition
                  - ":logs:"
                  - Ref: AWS::Region
                  - ":"
                  - Ref: AWS::AccountId
                  - :log-group:/aws/codebuild/
                  - Ref: BuildProject
            - Fn::Join:
                - ""
                - - "arn:"
                  - Ref: AWS::Partition
                  - ":logs:"
                  - Ref: AWS::Region
                  - ":"
                  - Ref: AWS::AccountId
                  - :log-group:/aws/codebuild/
                  - Ref: BuildProject
                  - :*
        - Action:
            - codebuild:CreateReportGroup
            - codebuild:CreateReport
            - codebuild:UpdateReport
            - codebuild:BatchPutTestCases
          Effect: Allow
          Resource:
            Fn::Join:
              - ""
              - - "arn:"
                - Ref: AWS::Partition
                - ":codebuild:"
                - Ref: AWS::Region
                - ":"
                - Ref: AWS::AccountId
                - :report-group/
                - Ref: BuildProject
                - -*
        - Action:
```

```
            - ecr:GetAuthorizationToken
          Effect: Allow
          Resource: "*"
        - Action:
            - ecr:BatchCheckLayerAvailability
            - ecr:CompleteLayerUpload
            - ecr:GetAuthorizationToken
            - ecr:InitiateLayerUpload
            - ecr:PutImage
            - ecr:UploadLayerPart
          Effect: Allow
          Resource:
            Fn::GetAtt:
              - ECRRepo
              - Arn
        - Action:
            - proton:GetService
          Effect: Allow
          Resource: "*"
        - Action:
            - s3:GetObject*
            - s3:GetBucket*
            - s3:List*
            - s3:DeleteObject*
            - s3:PutObject*
            - s3:Abort*
          Effect: Allow
          Resource:
            - Fn::GetAtt:
                - PipelineArtifactsBucket
                - Arn
            - Fn::Join:
                - ""
                - - Fn::GetAtt:
                      - PipelineArtifactsBucket
                      - Arn
                  - /*
        - Action:
            - kms:Decrypt
            - kms:DescribeKey
            - kms:Encrypt
            - kms:ReEncrypt*
            - kms:GenerateDataKey*
          Effect: Allow
```

```
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
          - Action:
              - kms:Decrypt
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
        Version: "2012-10-17"
      PolicyName: PublishRoleDefaultPolicy
      Roles:
        - Ref: PublishRole

  DeploymentRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              Service: codebuild.amazonaws.com
        Version: "2012-10-17"
  DeploymentRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - logs:CreateLogGroup
              - logs:CreateLogStream
              - logs:PutLogEvents
            Effect: Allow
            Resource:
              - Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
```

```
                - ":logs:"
                - Ref: AWS::Region
                - ":"
                - Ref: AWS::AccountId
                - :log-group:/aws/codebuild/Deploy*Project*
            - Fn::Join:
                - ""
                - - "arn:"
                  - Ref: AWS::Partition
                  - ":logs:"
                  - Ref: AWS::Region
                  - ":"
                  - Ref: AWS::AccountId
                  - :log-group:/aws/codebuild/Deploy*Project:*
        - Action:
            - codebuild:CreateReportGroup
            - codebuild:CreateReport
            - codebuild:UpdateReport
            - codebuild:BatchPutTestCases
          Effect: Allow
          Resource:
            Fn::Join:
              - ""
              - - "arn:"
                - Ref: AWS::Partition
                - ":codebuild:"
                - Ref: AWS::Region
                - ":"
                - Ref: AWS::AccountId
                - :report-group/Deploy*Project
                - -*
        - Action:
            - proton:UpdateServiceInstance
            - proton:GetServiceInstance
          Effect: Allow
          Resource: "*"
        - Action:
            - s3:GetObject*
            - s3:GetBucket*
            - s3:List*
          Effect: Allow
          Resource:
            - Fn::GetAtt:
                - PipelineArtifactsBucket
```

```
                - Arn
            - Fn::Join:
                - ""
                - - Fn::GetAtt:
                      - PipelineArtifactsBucket
                      - Arn
                  - /*
        - Action:
            - kms:Decrypt
            - kms:DescribeKey
          Effect: Allow
          Resource:
            Fn::GetAtt:
              - PipelineArtifactsBucketEncryptionKey
              - Arn
        - Action:
            - kms:Decrypt
            - kms:Encrypt
            - kms:ReEncrypt*
            - kms:GenerateDataKey*
          Effect: Allow
          Resource:
            Fn::GetAtt:
              - PipelineArtifactsBucketEncryptionKey
              - Arn
      Version: "2012-10-17"
    PolicyName: DeploymentRoleDefaultPolicy
    Roles:
      - Ref: DeploymentRole
PipelineArtifactsBucketEncryptionKey:
  Type: AWS::KMS::Key
  Properties:
    KeyPolicy:
      Statement:
        - Action:
            - kms:Create*
            - kms:Describe*
            - kms:Enable*
            - kms:List*
            - kms:Put*
            - kms:Update*
            - kms:Revoke*
            - kms:Disable*
            - kms:Get*
```

```
            - kms:Delete*
            - kms:ScheduleKeyDeletion
            - kms:CancelKeyDeletion
            - kms:GenerateDataKey
            - kms:TagResource
            - kms:UntagResource
          Effect: Allow
          Principal:
            AWS:
              Fn::Join:
                - ""
                - - "arn:"
                  - Ref: AWS::Partition
                  - ":iam::"
                  - Ref: AWS::AccountId
                  - :root
          Resource: "*"
        - Action:
            - kms:Decrypt
            - kms:DescribeKey
            - kms:Encrypt
            - kms:ReEncrypt*
            - kms:GenerateDataKey*
          Effect: Allow
          Principal:
            AWS:
              Fn::GetAtt:
                - PipelineRole
                - Arn
          Resource: "*"
        - Action:
            - kms:Decrypt
            - kms:DescribeKey
            - kms:Encrypt
            - kms:ReEncrypt*
            - kms:GenerateDataKey*
          Effect: Allow
          Principal:
            AWS:
              Fn::GetAtt:
                - PublishRole
                - Arn
          Resource: "*"
        - Action:
```

```
            - kms:Decrypt
            - kms:Encrypt
            - kms:ReEncrypt*
            - kms:GenerateDataKey*
          Effect: Allow
          Principal:
            AWS:
              Fn::GetAtt:
                - PublishRole
                - Arn
          Resource: "*"
        - Action:
            - kms:Decrypt
            - kms:DescribeKey
          Effect: Allow
          Principal:
            AWS:
              Fn::GetAtt:
                - DeploymentRole
                - Arn
          Resource: "*"
        - Action:
            - kms:Decrypt
            - kms:Encrypt
            - kms:ReEncrypt*
            - kms:GenerateDataKey*
          Effect: Allow
          Principal:
            AWS:
              Fn::GetAtt:
                - DeploymentRole
                - Arn
          Resource: "*"
      Version: "2012-10-17"
  UpdateReplacePolicy: Delete
  DeletionPolicy: Delete
PipelineArtifactsBucket:
  Type: AWS::S3::Bucket
  Properties:
    BucketEncryption:
      ServerSideEncryptionConfiguration:
        - ServerSideEncryptionByDefault:
            KMSMasterKeyID:
              Fn::GetAtt:
```

```
                - PipelineArtifactsBucketEncryptionKey
                - Arn
            SSEAlgorithm: aws:kms
      PublicAccessBlockConfiguration:
        BlockPublicAcls: true
        BlockPublicPolicy: true
        IgnorePublicAcls: true
        RestrictPublicBuckets: true
    UpdateReplacePolicy: Retain
    DeletionPolicy: Retain
  PipelineArtifactsBucketEncryptionKeyAlias:
    Type: AWS::KMS::Alias
    Properties:
      AliasName: 'alias/codepipeline-encryption-key-{{ service.name }}'     # resource
parameter
      TargetKeyId:
        Fn::GetAtt:
          - PipelineArtifactsBucketEncryptionKey
          - Arn
    UpdateReplacePolicy: Delete
    DeletionPolicy: Delete
  PipelineRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              Service: codepipeline.amazonaws.com
        Version: "2012-10-17"
  PipelineRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - s3:GetObject*
              - s3:GetBucket*
              - s3:List*
              - s3:DeleteObject*
              - s3:PutObject*
              - s3:Abort*
            Effect: Allow
```

```
            Resource:
              - Fn::GetAtt:
                  - PipelineArtifactsBucket
                  - Arn
              - Fn::Join:
                  - ""
                  - - Fn::GetAtt:
                        - PipelineArtifactsBucket
                        - Arn
                    - /*
          - Action:
              - kms:Decrypt
              - kms:DescribeKey
              - kms:Encrypt
              - kms:ReEncrypt*
              - kms:GenerateDataKey*
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineArtifactsBucketEncryptionKey
                - Arn
          - Action: codestar-connections:*
            Effect: Allow
            Resource: "*"
          - Action: sts:AssumeRole
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineBuildCodePipelineActionRole
                - Arn
          - Action: sts:AssumeRole
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - PipelineDeployCodePipelineActionRole
                - Arn
        Version: "2012-10-17"
      PolicyName: PipelineRoleDefaultPolicy
      Roles:
        - Ref: PipelineRole
  Pipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      RoleArn:
```

```
        Fn::GetAtt:
          - PipelineRole
          - Arn
      Stages:
        - Actions:
            - ActionTypeId:
                Category: Source
                Owner: AWS
                Provider: CodeStarSourceConnection
                Version: "1"
              Configuration:
                ConnectionArn: '{{ service.repository_connection_arn }}'   # resource
parameter
                FullRepositoryId: '{{ service.repository_id }}'            # resource
parameter
                BranchName: '{{ service.branch_name }}'                    # resource
parameter
              Name: Checkout
              OutputArtifacts:
                - Name: Artifact_Source_Checkout
              RunOrder: 1
          Name: Source
        - Actions:
            - ActionTypeId:
                Category: Build
                Owner: AWS
                Provider: CodeBuild
                Version: "1"
              Configuration:
                ProjectName:
                  Ref: BuildProject
              InputArtifacts:
                - Name: Artifact_Source_Checkout
              Name: Build
              OutputArtifacts:
                - Name: BuildOutput
              RoleArn:
                Fn::GetAtt:
                  - PipelineBuildCodePipelineActionRole
                  - Arn
              RunOrder: 1
          Name: Build {%- for service_instance in service_instances %}
        - Actions:
            - ActionTypeId:
```

```
                Category: Build
                Owner: AWS
                Provider: CodeBuild
                Version: "1"
              Configuration:
                ProjectName:
                  Ref: Deploy{{loop.index}}Project
              InputArtifacts:
                - Name: BuildOutput
              Name: Deploy
              RoleArn:
                Fn::GetAtt:
                  - PipelineDeployCodePipelineActionRole
                  - Arn
              RunOrder: 1
          Name: 'Deploy{{service_instance.name}}'         # resource parameter
{%- endfor %}
      ArtifactStore:
        EncryptionKey:
          Id:
            Fn::GetAtt:
              - PipelineArtifactsBucketEncryptionKey
              - Arn
          Type: KMS
        Location:
          Ref: PipelineArtifactsBucket
        Type: S3
    DependsOn:
      - PipelineRoleDefaultPolicy
      - PipelineRole
  PipelineBuildCodePipelineActionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              AWS:
                Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":iam::"
```

```
                  - Ref: AWS::AccountId
                  - :root
      Version: "2012-10-17"
  PipelineBuildCodePipelineActionRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
              - codebuild:BatchGetBuilds
              - codebuild:StartBuild
              - codebuild:StopBuild
            Effect: Allow
            Resource:
              Fn::GetAtt:
                - BuildProject
                - Arn
        Version: "2012-10-17"
      PolicyName: PipelineBuildCodePipelineActionRoleDefaultPolicy
      Roles:
        - Ref: PipelineBuildCodePipelineActionRole
  PipelineDeployCodePipelineActionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Statement:
          - Action: sts:AssumeRole
            Effect: Allow
            Principal:
              AWS:
                Fn::Join:
                  - ""
                  - - "arn:"
                    - Ref: AWS::Partition
                    - ":iam::"
                    - Ref: AWS::AccountId
                    - :root
        Version: "2012-10-17"
  PipelineDeployCodePipelineActionRoleDefaultPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyDocument:
        Statement:
          - Action:
```

```
              - codebuild:BatchGetBuilds
              - codebuild:StartBuild
              - codebuild:StopBuild
            Effect: Allow
            Resource:
              Fn::Join:
                - ""
                - - "arn:"
                  - Ref: AWS::Partition
                  - ":codebuild:"
                  - Ref: AWS::Region
                  - ":"
                  - Ref: AWS::AccountId
                  - ":project/Deploy*"
        Version: "2012-10-17"
      PolicyName: PipelineDeployCodePipelineActionRoleDefaultPolicy
      Roles:
        - Ref: PipelineDeployCodePipelineActionRole
Outputs:
  PipelineEndpoint:
    Description: The URL to access the pipeline
    Value: !Sub "https://${AWS::Region}.console.aws.amazon.com/codesuite/codepipeline/
pipelines/${Pipeline}/view?region=${AWS::Region}"
```

CodeBuild プロビジョニングテンプレートバンドル

CodeBuild プロビジョニングでは、IaC テンプレートを使用して IaC ファイルをレンダリングし、IaC プロビジョニングエンジンを使用して実行する代わりに、 AWS Proton シェルコマンドを単純に実行します。そのために、 は環境の AWS CodeBuild プロジェクトを環境アカウントで AWS Proton 作成し、 AWS Proton リソースの作成または更新ごとにコマンドを実行するジョブを開始します。テンプレートバンドルをオーサリングするとき、インフラストラクチャのプロビジョニングコマンドとデプロビジョニングコマンド、およびこれらのコマンドに必要なプログラム、スクリプト、その他のファイルを指定するマニフェストは、あなたが提供してください。あなたのコマンドでは、 AWS Proton が提供する入力を読み取ることができ、インフラのプロビジョニングまたはデプロビジョニングを行い、出力値を生成します。

マニフェスト AWS Proton は、コードが入力して入力値を取得できる入力ファイルを がレンダリングする方法も指定します。JSON または HCL にレンダリングできます。入力パラメータの詳細については、「the section called “CodeBuild プロビジョニングパラメータ”」を参照してください。マニフェストファイルについて詳しくは、「the section called “マニフェストとまとめ”」を参照してください。

> **Note**
>
> CodeBuild プロビジョニングは環境とサービスで使用できます。現時点では、この方法でコンポーネントをプロビジョニングすることはできません。

例: CodeBuild プロビジョニング AWS CDK で を使用する

CodeBuild プロビジョニングを使用する例として、 を使用して AWS リソース AWS Cloud Development Kit (AWS CDK) をプロビジョニング (デプロイ) およびプロビジョニング解除 (破棄) するコードと、CDK をインストールして CDK コードを実行するマニフェストを含めることができます。

以下のセクションでは、 AWS CDKで、プロビジョニングする CodeBuild プロビジョニングテンプレートバンドルに環境を組み込むことができるサンプルファイルの一覧を示します。

マニフェスト

次のマニフェストファイルは CodeBuild プロビジョニングを指定し、 のインストールと使用 AWS CDK、出力ファイル処理、および出力のレポートに必要なコマンドが含まれています AWS Proton。

Example infrastructure/manifest.yaml

```
infrastructure:
  templates:
    - rendering_engine: codebuild
      settings:
        image: aws/codebuild/amazonlinux2-x86_64-standard:4.0
        runtimes:
          nodejs: 16
        provision:
          - npm install
          - npm run build
          - npm run cdk bootstrap
          - npm run cdk deploy -- --require-approval never --outputs-file proton-
outputs.json
          - jq 'to_entries | map_values(.value) | add | to_entries | map({key:.key,
 valueString:.value})' < proton-outputs.json > outputs.json
          - aws proton notify-resource-deployment-status-change --resource-arn
 $RESOURCE_ARN --status IN_PROGRESS --outputs file://./outputs.json
        deprovision:
          - npm install
```

```
          - npm run build
          - npm run cdk destroy
        project_properties:
          VpcConfig:
            VpcId: "{{ environment.inputs.codebuild_vpc_id }}"
            Subnets: "{{ environment.inputs.codebuild_subnets }}"
            SecurityGroupIds: "{{ environment.inputs.codebuild_security_groups }}"
```

スキーマ

次のスキーマファイルは環境のパラメータを定義します。 AWS CDK コードは、デプロイ中にこれらのパラメータの値を参照できます。

Example schema/schema.yaml

```
schema:
  format:
    openapi: "3.0.0"
  environment_input_type: "MyEnvironmentInputType"
  types:
    MyEnvironmentInputType:
      type: object
      description: "Input properties for my environment"
      properties:
        my_sample_input:
          type: string
          description: "This is a sample input"
          default: "hello world"
        my_other_sample_input:
          type: string
          description: "Another sample input"
      required:
        - my_other_sample_input
```

AWS CDK ファイル

以下のファイルは Node.js CDK プロジェクトの例です。

Example infrastructure/package.json

```
{
  "name": "ProtonEnvironment",
  "version": "0.1.0",
```

```
  "bin": {
    "ProtonEnvironmente": "bin/ProtonEnvironment.js"
  },
  "scripts": {
    "build": "tsc",
    "watch": "tsc -w",
    "test": "jest",
    "cdk": "cdk"
  },
  "devDependencies": {
    "@types/jest": "^28.1.7",
    "@types/node": "18.7.6",
    "jest": "^28.1.3",
    "ts-jest": "^28.0.8",
    "aws-cdk": "2.37.1",
    "ts-node": "^10.9.1",
    "typescript": "~4.7.4"
  },
  "dependencies": {
    "aws-cdk-lib": "2.37.1",
    "constructs": "^10.1.77",
    "source-map-support": "^0.5.21"
  }
}
```

Example infrastructure/tsconfig.json

```
{
  "compilerOptions": {
    "target": "ES2018",
    "module": "commonjs",
    "lib": [
      "es2018"
    ],
    "declaration": true,
    "strict": true,
    "noImplicitAny": true,
    "strictNullChecks": true,
    "noImplicitThis": true,
    "alwaysStrict": true,
    "noUnusedLocals": false,
    "noUnusedParameters": false,
    "noImplicitReturns": true,
```

```
    "noFallthroughCasesInSwitch": false,
    "inlineSourceMap": true,
    "inlineSources": true,
    "experimentalDecorators": true,
    "strictPropertyInitialization": false,
    "resolveJsonModule": true,
    "esModuleInterop": true,
    "typeRoots": [
      "./node_modules/@types"
    ]
  },
  "exclude": [
    "node_modules",
    "cdk.out"
  ]
}
```

Example infrastructure/cdk.json

```
{
  "app": "npx ts-node --prefer-ts-exts bin/ProtonEnvironment.ts",
  "outputsFile": "proton-outputs.json",
  "watch": {
    "include": [
      "**"
    ],
    "exclude": [
      "README.md",
      "cdk*.json",
      "**/*.d.ts",
      "**/*.js",
      "tsconfig.json",
      "package*.json",
      "yarn.lock",
      "node_modules",
      "test"
    ]
  },
  "context": {
    "@aws-cdk/aws-apigateway:usagePlanKeyOrderInsensitiveId": true,
    "@aws-cdk/core:stackRelativeExports": true,
    "@aws-cdk/aws-rds:lowercaseDbIdentifier": true,
    "@aws-cdk/aws-lambda:recognizeVersionProps": true,
```

```
    "@aws-cdk/aws-cloudfront:defaultSecurityPolicyTLSv1.2_2021": true,
    "@aws-cdk-containers/ecs-service-extensions:enableDefaultLogDriver": true,
    "@aws-cdk/aws-ec2:uniqueImdsv2TemplateName": true,
    "@aws-cdk/core:target-partitions": [
      "aws",
      "aws-cn"
    ]
  }
}
```

Example infrastructure/bin/ProtonEnvironment.ts

```
#!/usr/bin/env node
import 'source-map-support/register';
import * as cdk from 'aws-cdk-lib';
import { ProtonEnvironmentStack } from '../lib/ProtonEnvironmentStack';

const app = new cdk.App();
new ProtonEnvironmentStack(app, 'ProtonEnvironmentStack', {});
```

Example infrastructure/lib/ProtonEnvironmentStack.ts

```
import { Stack, StackProps } from 'aws-cdk-lib';
import { Construct } from 'constructs';
import * as cdk from 'aws-cdk-lib';
import * as ssm from 'aws-cdk-lib/aws-ssm';
import input from '../proton-inputs.json';

export class ProtonEnvironmentStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, { ...props, stackName: process.env.STACK_NAME });

    const ssmParam = new ssm.StringParameter(this, "ssmParam", {
      stringValue: input.environment.inputs.my_sample_input,
      parameterName: `${process.env.STACK_NAME}-Param`,
      tier: ssm.ParameterTier.STANDARD
    })

    new cdk.CfnOutput(this, 'ssmParamOutput', {
      value: ssmParam.parameterName,
      description: 'The name of the ssm parameter',
      exportName: `${process.env.STACK_NAME}-Param`
    });
```

```
  }
}
```

レンダリング入力ファイル

CodeBuild ベースのプロビジョニングテンプレートで環境を作成すると、 AWS Proton では、指定した入力パラメータ値を含む入力ファイルがレンダリングされます。これらの値はあなたのコードで参照できます。次のファイルは、レンダリングされた入力ファイルの例です。

Example infrastructure/proton-inputs.json

```
{
  "environment": {
    "name": "myenv",
    "inputs": {
      "my_sample_input": "10.0.0.0/16",
      "my_other_sample_input": "11.0.0.0/16"
    }
  }
}
```

Terraform IaC ファイル

で Terraform Infrastructure as Code (IaC) ファイルを使用する方法について説明します AWS Proton。Terraform は、HashiCorp によって開発された、広く使用されているオープンソースの IaC エンジンです。Terraform モジュールは HashiCorp の HCL 言語で開発され、Amazon Web Services 含むいくつかのバックエンドインフラストラクチャプロバイダーをサポートしています。

AWS Proton は、Terraform IaC のセルフマネージドプロビジョニングをサポートしています。 IaC

プルリクエストに応答してインフラストラクチャプロビジョニングを実装するプロビジョニングリポジトリの完全な例については、GitHub 上のAWS Proton用の Terraform OpenSource GitHub Actions 自動化テンプレートを参照してください。

Terraform IaC テンプレートバンドルファイルにおけるセルフマネージドプロビジョニングの働き:

1. Terraform テンプレートバンドルから環境を作成すると、 はコンソールまたは`spec file`入力パラメータを使用して`.tf`ファイルを AWS Proton コンパイルします。
2. コンパイル済み IaC ファイルをマージするためのプルリクエストをAWS Protonに登録したリポジトリに送ります。

3. リクエストが承認されると、指定したプロビジョニングステータスを AWS Proton 待機します。
4. リクエストが拒否されると、環境の作成はキャンセルされます。
5. プルリクエストがタイムアウトになった場合、環境の作成は完了しません。

AWS Proton Terraform IaC に関する考慮事項:

- AWS Proton は Terraform プロビジョニングを管理しません。
- このリポジトリでプロビジョニングリポジトリを .makes プルリクエストに登録する必要があります。 AWS Proton AWS Proton
- プロビジョニングリポジトリ AWS Proton に接続するには、CodeStar 接続を作成する必要があります。
- AWS Proton コンパイルされた IaC ファイルからプロビジョニングするには、環境およびサービスの作成および更新アクション AWS Proton 後に AWS Proton プルリクエストに応答する必要があります。詳細については、「AWS Proton 環境」および「AWS Proton サービス」を参照してください。
- AWS Proton コンパイルされた IaC ファイルからパイプラインをプロビジョニングするには、CI/CD パイプラインリポジトリを作成する必要があります。
- プルリクエストベースのプロビジョニング自動化には、プロビジョニングされた AWS Proton リソースステータスの変更 AWS Proton を通知するステップが含まれている必要があります。 AWS Proton NotifyResourceDeploymentStatusChange API を使用できます。
- CloudFormation IaC ファイルから作成されたサービス、パイプライン、およびコンポーネントは、Terraform IaC ファイルから作成された環境にデプロイすることはできません。
- CloudFormation IaC ファイルから作成されたサービス、パイプライン、およびコンポーネントは、Terraform IaC ファイルから作成された環境にデプロイすることはできません。

Terraform IaC ファイルを準備するときは AWS Proton、次の例に示すように、入力変数に名前空間をアタッチします。詳細については、「パラメータ」を参照してください。

例 1: AWS Proton 環境 Terraform IaC ファイル

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.0"
```

```
    }
  }
  // This tells terraform to store the state file in s3 at the location
  // s3://terraform-state-bucket/tf-os-sample/terraform.tfstate
  backend "s3" {
    bucket = "terraform-state-bucket"
    key    = "tf-os-sample/terraform.tfstate"
    region = "us-east-1"
  }
}

// Configure the AWS Provider
provider "aws" {
  region = "us-east-1"
  default_tags {
    tags = var.proton_tags
  }
}

resource "aws_ssm_parameter" "my_ssm_parameter" {
  name  = "my_ssm_parameter"
  type  = "String"
  // Use the Proton environment.inputs. namespace
  value = var.environment.inputs.ssm_parameter_value
}
```

コンパイル済み Infrastructure as Code として

環境またはサービスを作成すると、 はコンソールまたはspec file入力を使用してインフラストラクチャをコードファイルとして AWS Proton コンパイルします。これで、次の例に示すように、Terraform であなたの入力に使用できる proton.*resource-type*.variables.tf ファイルと proton.auto.tfvars.json ファイルが作成されます。これらのファイルは、環境名やサービスインスタンス名と同じフォルダ内の指定されたリポジトリにあります。

この例では、 が変数定義と変数値にタグ AWS Proton を含める方法と、これらの AWS Proton タグをプロビジョニングされたリソースに伝達する方法を示しています。詳細については、「the section called “プロビジョニングされたリソースへのタグの伝達”」を参照してください。

例 2:「dev」という環境向けにコンパイルされた IaC ファイル。

dev/environment.tf:

```
terraform {
```

```
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
  // This tells terraform to store the state file in s3 at the location
  // s3://terraform-state-bucket/tf-os-sample/terraform.tfstate
  backend "s3" {
    bucket = "terraform-state-bucket"
    key    = "tf-os-sample/terraform.tfstate"
    region = "us-east-1"
  }
}

// Configure the AWS Provider
provider "aws" {
  region = "us-east-1"
  default_tags {
    tags = var.proton_tags
  }
}

resource "aws_ssm_parameter" "my_ssm_parameter" {
  name  = "my_ssm_parameter"
  type  = "String"
  // Use the Proton environment.inputs. namespace
  value = var.environment.inputs.ssm_parameter_value
}
```

dev/proton.environment.variables.tf:

```
variable "environment" {
  type = object({
    inputs = map(string)
    name = string
  })
}

variable "proton_tags" {
  type = map(string)
  default = null
}
```

dev/proton.auto.tfvars.json:

```
{
  "environment": {
    "name": "dev",
    "inputs": {
      "ssm_parameter_value": "MyNewParamValue"
    }
  }

  "proton_tags" : {
    "proton:account" : "123456789012",
    "proton:template" : "arn:aws:proton:us-east-1:123456789012:environment-template/
fargate-env",
    "proton:environment" : "arn:aws:proton:us-east-1:123456789012:environment/dev"
  }
}
```

リポジトリパス

AWS Proton は、環境またはサービス作成アクションからのコンソールまたは仕様入力を使用して、コンパイルされた IaC ファイルを見つけるリポジトリとパスを見つけます。入力値は、名前空間入力パラメータに渡されます。

AWS Proton は 2 つのリポジトリパスレイアウトをサポートしています。次の例では、2 つの環境の名前空間リソースパラメータによってパスの名前が指定されています。各環境には 2 つのサービスのサービスインスタンスがあり、そのうちの 1 つのサービスのサービスインスタンスにはコンポーネントが直接定義されています。

リソースタイプ	名前パラメータ	=	リソース名
環境	environment.name		"env-prod"
環境	environment.name	=	"env-staged"
サービス	service.name		"service-one"

リソースタイプ	名前パラメータ	=	リソース名
サービスインスタンス	service_instance.name		"instance-one-prod"
サービスインスタンス	service_instance.name		"instance-one-staged"
サービス	service.name		"service-two"
サービスインスタンス	service_instance.name		"instance-two-prod"
コンポーネント	service_instance.components.default.name		"component-prod"
サービスインスタンス	service_instance.name		"instance-two-staged"
コンポーネント	service_instance.components.default.name		"component-staged"

Layout 1

が environmentsフォルダを持つ指定されたリポジトリ AWS Proton を見つけると、コンパイルされた IaC ファイルを含むフォルダが作成され、 で名前が付けられますenvironment.name。

が、サービスインスタンス互換環境名と一致するフォルダ名を含む environments フォルダを持つ指定されたリポジトリ AWS Proton を見つけると、コンパイルされたインスタンス IaC ファイルを含むフォルダが作成され、 という名前が付けられますservice_instance.name。

```
/repo
```

```
/environments
    /env-prod                              # environment folder
        main.tf
        proton.environment.variables.tf
        proton.auto.tfvars.json

        /service-one-instance-one-prod     # instance folder
            main.tf
            proton.service_instance.variables.tf
            proton.auto.tfvars.json

        /service-two-instance-two-prod     # instance folder
            main.tf
            proton.service_instance.variables.tf
            proton.auto.tfvars.json

        /component-prod                    # component folder
            main.tf
            proton.component.variables.tf
            proton.auto.tfvars.json

    /env-staged                            # environment folder
        main.tf
        proton.variables.tf
        proton.auto.tfvars.json

        /service-one-instance-one-staged   # instance folder
            main.tf
            proton.service_instance.variables.tf
            proton.auto.tfvars.json

        /service-two-instance-two-staged   # instance folder
            main.tf
            proton.service_instance.variables.tf
            proton.auto.tfvars.json

        /component-staged                  # component folder
            main.tf
            proton.component.variables.tf
            proton.auto.tfvars.json
```

Layout 2

がフォルダなしで指定されたリポジトリ AWS Proton を検出するとenvironments、コンパイルされた環境 IaC ファイルがあるenvironment.nameフォルダが作成されます。

がサービスインスタンス互換環境名に一致するフォルダ名を持つ指定されたリポジトリ AWS Proton を検出すると、コンパイルされたインスタンス IaC ファイルがあるservice_instance.nameフォルダが作成されます。

```
/repo
    /env-prod                           # environment folder
        main.tf
        proton.environment.variables.tf
        proton.auto.tfvars.json

        /service-one-instance-one-prod    # instance folder
            main.tf
            proton.service_instance.variables.tf
            proton.auto.tfvars.json

        /service-two-instance-two-prod    # instance folder
            main.tf
            proton.service_instance.variables.tf
            proton.auto.tfvars.json

        /component-prod                   # component folder
            main.tf
            proton.component.variables.tf
            proton.auto.tfvars.json

    /env-staged                           # environment folder
        main.tf
        proton.variables.tf
        proton.auto.tfvars.json

        /service-one-instance-one-staged  # instance folder
            main.tf
            proton.service_instance.variables.tf
            proton.auto.tfvars.json

        /service-two-instance-two-staged  # instance folder
            main.tf
            proton.service_instance.variables.tf
```

```
        proton.auto.tfvars.json

    /component-staged                # component folder
        main.tf
        proton.component.variables.tf
        proton.auto.tfvars.json
```

スキーマファイル

管理者は、Open API Data Models (スキーマ) セクションを使用してテンプレートバンドルのパラメータスキーマ YAML ファイルを定義すると、スキーマで定義した要件に照らしてパラメータ値の入力 AWS Proton を検証できます。

形式および使用可能なキーワードの詳細については、OpenAPI のスキーマオブジェクトセクションを参照してください。

環境テンプレートバンドルのスキーマ要件

スキーマは、YAML 形式の OpenAPI のデータモデル (スキーマ)セクションに従う必要があります。また、環境テンプレートバンドルの一部でなければなりません。

環境スキーマの場合、Open API のデータモデル (スキーマ) セクションに従っていることを立証するために、書式付きヘッダーを含める必要があります。次の環境スキーマの例では、これらのヘッダーが先頭の 3 行に表示されます。

`environment_input_type` を含めて所与の名前で定義する必要があります。次の例では、5 行目で定義されています。このパラメータを定義することで、 AWS Proton 環境リソースに関連付けます。

Open API スキーマモデルに従うには、`types` を含める必要があります。次の例では、6 行目が該当します。

`types` に続いて、`environment_input_type` タイプを定義する必要があります。環境の入力パラメータを `environment_input_type` のプロパティとして定義します。スキーマに関連付けられた環境 Infrastructure as Code (IaC) ファイル内にリストのある 1 つ以上のパラメータと一致する名前のプロパティを 1 つ以上含める必要があります。

環境を作成し、カスタマイズされたパラメータ値を指定すると、 はスキーマファイル AWS Proton を使用して、関連する CloudFormation IaC ファイル内の中括弧で囲まれたパラメータを照合、

検証、挿入します。プロパティ (パラメータ) ごとに name と type を指定します。オプションで、description、default、pattern を指定します。

次の標準環境スキーマの例に示す定義済みパラメータには、vpc_cidr、subnet_one_cidr、および subnet_two_cidr があり、default キーワードとデフォルト値が与えられています。この環境テンプレートバンドルスキーマを使用して環境を作成する際には、デフォルト値を受け入れるか、または独自の値を指定できます。パラメータが required プロパティ (パラメータ) としてリストに示され、デフォルト値がない場合、環境の作成時に値を入力する必要があります。

2 つ目の例の標準環境テンプレートスキーマには required パラメータ my_other_sample_input のリストがあります。

2 種類の環境テンプレートのスキーマを作成できます。詳細については、「テンプレートを登録してパブリッシュする」を参照してください。

- 標準環境テンプレート

  次の例では、説明と入力プロパティを使用して環境入力タイプを定義します。このスキーマの例は、例 3 に示す AWS Proton CloudFormation IaC ファイルと組み合わせて使用できます。

  標準環境テンプレートのスキーマの例:

```
schema:                             # required
  format:                           # required
    openapi: "3.0.0"                # required
  # required              defined by administrator
  environment_input_type: "PublicEnvironmentInput"
  types:                            # required
    # defined by administrator
    PublicEnvironmentInput:
      type: object
      description: "Input properties for my environment"
      properties:
        vpc_cidr:                    # parameter
          type: string
          description: "This CIDR range for your VPC"
          default: 10.0.0.0/16
          pattern: ([0-9]{1,3}\.){3}[0-9]{1,3}($|/(16|24))
        subnet_one_cidr:             # parameter
          type: string
          description: "The CIDR range for subnet one"
          default: 10.0.0.0/24
```

```
      pattern: ([0-9]{1,3}\.){3}[0-9]{1,3}($|/(16|24))
    subnet_two_cidr:            # parameter
      type: string
      description: "The CIDR range for subnet one"
      default: 10.0.1.0/24
      pattern: ([0-9]{1,3}\.){3}[0-9]{1,3}($|/(16|24))
```

required パラメータを含む標準環境テンプレートのスキーマの例:

```
schema:                           # required
  format:                         # required
    openapi: "3.0.0"              # required
  # required              defined by administrator
  environment_input_type: "MyEnvironmentInputType"
  types:                          # required
    # defined by administrator
    MyEnvironmentInputType:
      type: object
      description: "Input properties for my environment"
      properties:
        my_sample_input:          # parameter
          type: string
          description: "This is a sample input"
          default: "hello world"
        my_other_sample_input:    # parameter
          type: string
          description: "Another sample input"
        another_optional_input:   # parameter
          type: string
          description: "Another optional input"
          default: "!"
      required:
        - my_other_sample_input
```

- カスタマーマネージド環境テンプレート

  次の例では、スキーマには、カスタマーマネージドインフラストラクチャのプロビジョニングに使用した IaC からの出力を複製する出力リストのみが含まれています。出力値の型を (列挙、配列、その他ではなく) 文字列のみとして定義する必要があります。たとえば、次のコードスニペットは、外部 CloudFormation テンプレートの出力セクションを示しています。これは、例 1 に示すテンプレートからとったものです。これは、例 4 から作成された AWS Proton Fargate サービスの外部カスタマーマネージドインフラストラクチャを作成するために使用できます。

> **Important**
>
> 管理者は、プロビジョニングおよびマネージドインフラストラクチャとすべての出力パラメータが、関連するカスタマーマネージド環境テンプレートと互換性があることを確認する必要があります。これらの変更は表示されないため、 はユーザーに代わって変更を考慮 AWS Proton できません AWS Proton。一貫性がない場合、結果はエラーになります。

カスタマーマネージド環境テンプレートの CloudFormation の IaC ファイル出力例:

```
// Cloudformation Template Outputs
[...]
Outputs:
  ClusterName:
    Description: The name of the ECS cluster
    Value: !Ref 'ECSCluster'
  ECSTaskExecutionRole:
    Description: The ARN of the ECS role
    Value: !GetAtt 'ECSTaskExecutionRole.Arn'
  VpcId:
    Description: The ID of the VPC that this stack is deployed in
    Value: !Ref 'VPC'
[...]
```

対応する AWS Proton カスタマーマネージド環境テンプレートバンドルのスキーマを次の例に示します。各出力値は文字列として定義されます。

カスタマーマネージド環境テンプレートのスキーマの例:

```
schema:                             # required
  format:                           # required
    openapi: "3.0.0"                # required
  # required               defined by administrator
  environment_input_type: "EnvironmentOutput"
  types:                            # required
    # defined by administrator
    EnvironmentOutput:
      type: object
      description: "Outputs of the environment"
      properties:
```

```
        ClusterName:               # parameter
          type: string
          description: "The name of the ECS cluster"
        ECSTaskExecutionRole:      # parameter
          type: string
          description: "The ARN of the ECS role"
        VpcId:                     # parameter
          type: string
          description: "The ID of the VPC that this stack is deployed in"
[...]
```

サービステンプレートバンドルのスキーマ要件

次の例に示すように、スキーマは YAML 形式の OpenAPI のデータモデル (スキーマ)セクションに従う必要があります。サービステンプレートバンドルでスキーマファイルを指定する必要があります。

次のサービススキーマの例では、書式付きヘッダーを含める必要があります。次の例では、先頭の 3 行が該当します。これは、Open API のデータモデル (スキーマ) セクションに従っていることを立証するためです。

`service_input_type` を含めて所与の名前で定義する必要があります。次の例では、5 行目が該当します。これにより、パラメータが AWS Proton サービスリソースに関連付けられます。

コンソールまたは CLI を使用して AWS Proton サービスを作成する場合、サービスパイプラインはデフォルトで含まれます。サービスにサービスパイプラインを含めるには、名前を指定して `pipeline_input_type` を含める必要があります。次の例では、7 行目が該当します。 AWS Proton サービスパイプラインを含めない場合は、このパラメータを含めないでください。詳細については、「テンプレートを登録してパブリッシュする」を参照してください。

Open API スキーマモデルに従うには、`types` を含める必要があります。次の例では、9 行目が該当します。

`types` に続いて、`service_input_type` タイプを定義する必要があります。サービスの入力パラメータを `service_input_type` のプロパティとして定義します。スキーマに関連付けられたサービス Infrastructure as Code (IaC) ファイル内にリストのある 1 つ以上のパラメータと一致する名前のプロパティを 1 つ以上含める必要があります。

サービスパイプラインを定義するには、`service_input_type` 定義の下で `pipeline_input_type` を定義する必要があります。上記のように、スキーマに関連付けられた IaC ファイル内にリストのある 1 つ以上のパラメータと一致する名前のプロパティを 1 つ以上含め

る必要があります。 AWS Proton サービスパイプラインを含めない場合は、この定義を含めないでください。

管理者または開発者としてサービスを作成し、カスタマイズされたパラメータ値を指定すると、 はスキーマファイル AWS Proton を使用して、関連する CloudFormation IaC ファイルの中括弧で囲まれたパラメータを照合、検証、挿入します。プロパティ (パラメータ) ごとに `name` と `type` を指定します。オプションで、`description`、`default`、`pattern` も指定します。

スキーマの例に示す定義済みパラメータは、`port`、`desired_count`、`task_size` および `image` で `default` キーワードとデフォルト値が付いています。このサービステンプレートバンドルスキーマを使用してサービスを作成する際には、デフォルト値を受け入れるか、または独自の値を指定できます。パラメータ `unique_name` は例に含まれ、デフォルト値はありません。リストには `required` プロパティ (パラメータ) として表示されます。管理者または開発者としてのあなたは、サービスの作成時に `required` パラメータの値を指定する必要があります。

サービスパイプライン付きでサービステンプレートを作成しようとする場合、スキーマに `pipeline_input_type` を含めます。

サービスパイプラインを含むサービスの AWS Proton サービススキーマファイルの例。

このスキーマ例は、例 4 および例 5 に示す AWS Proton IaC ファイルで使用できます。サービスパイプラインが含まれています。

```
schema:                             # required
  format:                           # required
    openapi: "3.0.0"                # required
  # required           defined by administrator
  service_input_type: "LoadBalancedServiceInput"
  # only include if including AWS Proton service pipeline, defined by administrator
  pipeline_input_type: "PipelineInputs"

  types:                            # required
    # defined by administrator
    LoadBalancedServiceInput:
      type: object
      description: "Input properties for a loadbalanced Fargate service"
      properties:
        port:                       # parameter
          type: number
          description: "The port to route traffic to"
          default: 80
          minimum: 0
```

```
        maximum: 65535
      desired_count:             # parameter
        type: number
        description: "The default number of Fargate tasks you want running"
        default: 1
        minimum: 1
      task_size:                 # parameter
        type: string
        description: "The size of the task you want to run"
        enum: ["x-small", "small", "medium", "large", "x-large"]
        default: "x-small"
      image:                     # parameter
        type: string
        description: "The name/url of the container image"
        default: "public.ecr.aws/z9d2n7e1/nginx:1.19.5"
        minLength: 1
        maxLength: 200
      unique_name:               # parameter
        type: string
        description: "The unique name of your service identifier. This will be used
to name your log group, task definition and ECS service"
        minLength: 1
        maxLength: 100
    required:
      - unique_name
  # defined by administrator
  PipelineInputs:
    type: object
    description: "Pipeline input properties"
    properties:
      dockerfile:                # parameter
        type: string
        description: "The location of the Dockerfile to build"
        default: "Dockerfile"
        minLength: 1
        maxLength: 100
      unit_test_command:         # parameter
        type: string
        description: "The command to run to unit test the application code"
        default: "echo 'add your unit test command here'"
        minLength: 1
        maxLength: 200
```

サービスパイプラインなしでサービステンプレートを作成しようとするあなたのコンポーネントる場合、次の例に示すように、スキーマに pipeline_input_type を含めません。

サービスパイプラインを含まないサービスの AWS Proton サービススキーマファイルの例

```
schema:                           # required
  format:                         # required
    openapi: "3.0.0"              # required
  # required            defined by administrator
  service_input_type: "MyServiceInstanceInputType"

  types:                          # required
    # defined by administrator
    MyServiceInstanceInputType:
      type: object
      description: "Service instance input properties"
      required:
        - my_sample_service_instance_required_input
      properties:
        my_sample_service_instance_optional_input:   # parameter
          type: string
          description: "This is a sample input"
          default: "hello world"
        my_sample_service_instance_required_input:   # parameter
          type: string
          description: "Another sample input"
```

のテンプレートファイルをまとめる AWS Proton

環境とサービスの Infrastructure as Code (IaC) ファイルとそれぞれのスキーマファイルを準備したら、それらをディレクトリに編成する必要があります。マニフェスト YAML ファイルも作成する必要があります。マニフェストファイルには、ディレクトリ内の IaC ファイル、レンダリングエンジン、およびこのテンプレート内の IaC の開発に使用されたテンプレート言語が記載されています。

> **Note**
>
> マニフェストファイルは、テンプレートバンドルとは別に、直接定義されたコンポーネントへの直接入力として使用することもできます。この場合、CloudFormation と Terraform の両方について、常に単一の IaC テンプレートファイルが指定されます。コンポーネントの詳細については、「コンポーネント」を参照してください。

マニフェストファイルは、次の例に示す形式と内容に従う必要があります。

CloudFormation マニフェストファイル形式:

CloudFormation では、リストに単一のファイルを含めます。

```
infrastructure:
  templates:
    - file: "cloudformation.yaml"
      rendering_engine: jinja
      template_language: cloudformation
```

Terraform マニフェストファイル形式:

terraform では、1 つのファイルを明示的にリストに入れるか、またはワイルドカード * でディレクトリ内の各ファイルをリストに加えます。

> **Note**
>
> ワイルドカードには、名前が .tf で終わるファイルのみが含まれます。他のファイルは無視されます。

```
infrastructure:
  templates:
    - file: "*"
      rendering_engine: hcl
      template_language: terraform
```

CodeBuild ベースのプロビジョニングマニフェストファイル形式:

CodeBuild ベースのプロビジョニングでは、プロビジョニングシェルコマンドとデプロビジョニングシェルコマンドを指定します。

> **Note**
>
> マニフェストだけでなく、バンドルにもあなたのコマンドが依存するすべてのファイルを含めておく必要があります。

次のマニフェスト例では、CodeBuild ベースのプロビジョニングを使用して、 AWS Cloud Development Kit (AWS CDK) () を使用してリソースをプロビジョニング (デプロイ) およびプロビジョニング解除 (破棄) しますAWS CDK。テンプレートバンドルには CDK コードも含めておく必要があります。

プロビジョニング中に、AWS Protonは、テンプレートのスキーマに名前 `proton-input.json` で定義した入力パラメータの値を含む入力ファイルを作成します。

```
infrastructure:
  templates:
    - rendering_engine: codebuild
      settings:
        image: aws/codebuild/amazonlinux2-x86_64-standard:4.0
        runtimes:
          nodejs: 16
        provision:
          - npm install
          - npm run build
          - npm run cdk bootstrap
          - npm run cdk deploy -- --require-approval never --outputs-file proton-
outputs.json
          - jq 'to_entries | map_values(.value) | add | to_entries | map({key:.key,
 valueString:.value})' < proton-outputs.json > outputs.json
          - aws proton notify-resource-deployment-status-change --resource-arn
 $RESOURCE_ARN --status IN_PROGRESS --outputs file://./outputs.json
        deprovision:
          - npm install
          - npm run build
          - npm run cdk destroy
        project_properties:
          VpcConfig:
            VpcId: "{{ environment.inputs.codebuild_vpc_id }}"
            Subnets: "{{ environment.inputs.codebuild_subnets }}"
            SecurityGroupIds: "{{ environment.inputs.codebuild_security_groups }}"
```

環境またはサービステンプレートバンドルのディレクトリとマニフェストファイルをセットアップしたら、ディレクトリを tar ボールに gzip して、 が取得 AWS Proton できる Amazon Simple Storage Service (Amazon S3) バケット、またはテンプレート同期 Git リポジトリにアップロードします。

登録した環境またはサービステンプレートのマイナーバージョンを作成するときは AWS Proton、S3 バケットにある環境またはサービステンプレートバンドル tar ball へのパスを指定します。 AWS

Proton は、新しいテンプレートのマイナーバージョンで保存します。新しいテンプレートのマイナーバージョンを選択して、環境またはサービスを作成または更新できます AWS Proton。

環境テンプレートバンドルのまとめ

作成する環境テンプレートバンドルには 2 つのタイプがあります AWS Proton。

- 標準環境テンプレートの環境テンプレートバンドルを作成するには、以下の環境テンプレートバンドルのディレクトリ構造のように、スキーマ、Infrastructure as Code (IaC) ファイル、マニフェストファイルをディレクトリに編成します。
- カスタマーマネージド環境テンプレートの環境テンプレートバンドルを作成するには。スキーマファイルとディレクトリのみを指定します。インフラストラクチャディレクトリとファイルを含めないでください。インフラストラクチャディレクトリとファイルが含まれている場合 AWS Proton 、エラーが発生します。

詳細については、「テンプレートを登録してパブリッシュする」を参照してください。

CloudFormation 環境テンプレートバンドルディレクトリ構造:

```
/schema
  schema.yaml
/infrastructure
  manifest.yaml
  cloudformation.yaml
```

Terraform 環境テンプレートバンドルディレクトリ構造:

```
/schema
  schema.yaml
/infrastructure
  manifest.yaml
  environment.tf
```

サービステンプレートバンドルをまとめる

サービステンプレートバンドルを作成するには、サービステンプレートバンドルのディレクトリ構造例に示すように、スキーマ、Infrastructure as Code (IaC) ファイル、およびマニフェストファイルをディレクトリに編成する必要があります。

あなたのテンプレートバンドルにサービスパイプラインを含めない場合は、このテンプレートバンドルに関連付けるサービステンプレートを作成するときに、パイプラインディレクトリとファイルを含めず、`"pipelineProvisioning": "CUSTOMER_MANAGED"` を設定してください。

> **Note**
>
> サービステンプレートの作成後は、`pipelineProvisioning` を変更できません。

詳細については、「テンプレートを登録してパブリッシュする」を参照してください。

CloudFormation サービステンプレートバンドルディレクトリ構造:

```
/schema
  schema.yaml
/instance_infrastructure
  manifest.yaml
  cloudformation.yaml
/pipeline_infrastructure
  manifest.yaml
  cloudformation.yaml
```

Terraform サービステンプレートバンドルディレクトリ構造:

```
/schema
  schema.yaml
/instance_infrastructure
  manifest.yaml
  instance.tf
/pipeline_infrastructure
  manifest.yaml
  pipeline.tf
```

テンプレートバンドルに関する考慮事項

- Infrastructure as code (IaC) ファイル

  AWS Proton は、テンプレートの正しいファイル形式を監査します。ただし、テンプレート開発、依存関係、ロジックエラーはチェック AWS Proton されません。例えば、サービスまたは環境テ

ンプレートの一部として、 CloudFormation IaC ファイルに Amazon S3 バケットの作成を指定したとします。これらのテンプレートに基づいてサービスが作成されます。さて、ある時点でサービスを削除したいとします。指定された S3 バケットが空ではなく、CloudFormation IaC ファイルが `Retain`で としてマークしない場合`DeletionPolicy`、 はサービス削除オペレーションで AWS Proton 失敗します。

- バンドルファイルサイズの制限および形式
  - バンドルファイルのサイズ、数、および名前サイズの制限は、AWS Proton クォータ で確認できます。
  - ファイルのテンプレートバンドルディレクトリは、gzip で圧縮した tar ball として Amazon Simple Storage Service (Amazon S3) バケット内にあります。
  - バンドル内の各ファイルは、有効な形式の YAML ファイルでなければなりません。
- S3 バケットテンプレートバンドル暗号化

  S3 バケットに保存されているテンプレートバンドル内の機密データを暗号化する場合は、SSE-S3 または SSE-KMS キーを使用して、 AWS Proton がそれらを取得できるようにします。

AWS Proton テンプレート

テンプレートバンドルを AWS Proton テンプレートライブラリに追加するには、テンプレートのマイナーバージョンを作成して に登録します AWS Proton。テンプレートを作成する際に、Amazon S3 バケットの名前とあなたのテンプレートバンドルのパスを指定します。テンプレートをパブリッシュした後、プラットフォームチームのメンバーや開発者がそれらを選択できます。選択すると、はテンプレート AWS Proton を使用してインフラストラクチャとアプリケーションを作成およびプロビジョニングします。

管理者は、 環境テンプレートを作成して登録できます AWS Proton。この環境テンプレートを使用して、複数の環境をデプロイできます。たとえば、「dev」、「staging」、「prod」の環境のデプロイに使用できます。「dev」環境には、プライベートサブネットを持つ VPC と、すべてのリソースへの制限付きアクセスポリシーが含まれる可能性があります。環境出力をサービスの入力として使用できます。

環境テンプレートを作成して登録し、異なる 2 種類の環境を作成できます。ユーザーと開発者の両方が、両方のタイプ AWS Proton にサービスをデプロイするために使用できます。

- AWS Proton が環境インフラストラクチャをプロビジョニングして管理する標準環境を作成するために使用する標準環境テンプレートを登録して公開します。
- AWS Proton が既存のプロビジョニング済みインフラストラクチャに接続するカスタマーマネージド環境を作成するために使用するカスタマーマネージド環境テンプレートを登録して公開します。AWS Proton は、既存のプロビジョニング済みインフラストラクチャを管理しません。

サービステンプレートを作成して に登録 AWS Proton し、 環境にサービスをデプロイできます。サービスをデプロイする前に、 AWS Proton 環境を作成する必要があります。

次のリストは、 でテンプレートを作成および管理する方法を示しています AWS Proton。

- (オプション) AWS Proton API コールと IAM サービスロールへの開発者アクセスを制御する IAM AWS Proton ロールを準備します。詳細については、「the section called “IAM ロール”」を参照してください。
- テンプレートバンドルを構成します。詳細については、「テンプレートバンドル」を参照してください。
- テンプレートバンドルが構成、圧縮され、Amazon S3 バケットに保存された AWS Proton 後、テンプレートを作成して に登録します。これはコンソールまたは AWS CLIを使用して実行できます。

- テンプレートをテストして使用し、登録後に AWS Proton プロビジョニングされたリソースを作成および管理します AWS Proton。
- テンプレートのライフサイクル全体を通して、メジャーバージョンとマイナーバージョンを作成および管理します。

テンプレートのバージョンは、手動で管理するか、またはテンプレート同期設定を使用して管理できます。

- AWS Proton コンソールと を使用して AWS CLI 、新しいマイナーバージョンまたはメジャーバージョンを作成します。
- 定義したリポジトリ内のテンプレートバンドルへの変更を検出すると、 が新しいマイナーバージョンまたはメジャーバージョンを自動的に作成できるようにするテンプレート同期設定を作成します。 AWS Proton

詳細については、「 AWS Proton サービス API リファレンス」を参照してください。

トピック

バージョン付きテンプレート

管理者またはプラットフォームチームのメンバーとして、インフラストラクチャリソースのプロビジョニングに使用されるバージョン対応テンプレートのライブラリを定義、作成、および管理します。テンプレートバージョンには、マイナーバージョンとメジャーバージョンの 2 種類があります。

- マイナーバージョン — 下位互換性があるスキーマを含むテンプレートへの変更。これらの変更では、新しいテンプレートバージョンへの更新時に開発者が新しい情報を提供する必要がありません。

マイナーバージョンの変更を試みると、 AWS Proton は、新しいバージョンのスキーマがテンプレートの以前のマイナーバージョンと下位互換性があるかどうかをベストエフォートで判断しようとします。新しいスキーマに下位互換性がない場合、 は新しいマイナーバージョンの登録に AWS Proton 失敗します。

> **Note**
>
> 互換性は、chema. AWS Proton does のみに基づいて決定されます。テンプレートバンドルのInfrastructure as Code (IaC) ファイルが以前のマイナーバージョンと下位互換性があるかどうかはチェックされません。例えば、新しい IaC ファイルが、以前のマイナーバージョンのテンプレートによってプロビジョニングされたインフラストラクチャで実行されているアプリケーションに重大な変更を引き起こすかどうかはチェック AWS Proton しません。

- メジャーバージョン — 下位互換性がないおそれのある、テンプレートに加えられた変更。そのような変更には、通常、開発者からの新しい入力が必要です。また、多くの場合、テンプレートスキーマの変更も必要です。

  場合によっては、下位互換性のある変更を、あなたのチームの運用モデルに基づいたメジャーバージョンとして指定することも選択できます。

テンプレートバージョンリクエストがマイナーバージョンかメジャーバージョンか AWS Proton を決定する方法は、テンプレートの変更の追跡方法によって異なります。

- 新しいテンプレートバージョンの作成を明示的にリクエストするとき、メジャーバージョンはメジャーバージョン番号を指定してリクエストし、マイナーバージョンはメジャーバージョン番号を指定せずにリクエストします。
- テンプレート同期を使用する (したがって、明示的なテンプレートバージョンリクエストを行わない) 場合、 は、既存の YAML ファイルで発生するテンプレート変更の新しいマイナーバージョンを作成 AWS Proton しようとします。新しいテンプレート変更の新しいディレクトリを作成すると (例えば、v1 から v2 への移行) 、 はメジャーバージョン AWS Proton を作成します。

> **Note**
>
> が変更に下位互換性がない AWS Proton と判断した場合、テンプレート同期に基づく新しいマイナーバージョン登録は失敗します。

テンプレートの新しいバージョンをパブリッシュした場合、そのメジャーバージョンとマイナーバージョンが最も高ければ、それが推奨バージョンになります。新しい AWS Proton リソースは新しい推奨バージョンを使用して作成され、管理者は新しいバージョンを使用し、古いバージョンを使用している既存の AWS Proton リソースを更新するように AWS Proton 求められます。

テンプレートを登録してパブリッシュする

以下のセクションで説明するように AWS Proton、 で環境テンプレートとサービステンプレートを登録して公開できます。

コンソールまたは を使用して、テンプレートの新しいバージョンを作成できます AWS CLI。

または、 コンソールまたは AWS CLI を使用してテンプレートを作成し、テンプレート同期を設定することもできます。この設定では、定義した登録済み git リポジトリにあるテンプレートバンドルから AWS Proton 同期できます。あなたのテンプレートバンドルのいずれかを変更するコミットがあなたのリポジトリにプッシュされるたびに、あなたのテンプレートの新しいマイナーバージョンまたはメジャーバージョンが (まだ存在しなければ) 作成されます。テンプレート同期設定の前提条件と要件の詳細については、「テンプレートの同期設定」を参照してください。

環境テンプレートの登録とパブリッシュ

次のタイプの環境テンプレートを登録してパブリッシュできます。

- が環境インフラストラクチャのデプロイと管理 AWS Proton に使用する標準環境テンプレートを登録して公開します。
- が管理する既存のプロビジョニング済みインフラストラクチャに接続 AWS Proton するために使用するカスタマーマネージド環境テンプレートを登録して公開します。 AWS Proton は、既存のプロビジョニング済みインフラストラクチャを管理しません。

> **Important**
>
> 管理者として、プロビジョニングおよびマネージドインフラストラクチャとすべての出力パラメータが、関連するカスタマーマネージド環境テンプレートと互換性があることを確認してください。 は、ユーザーに代わって変更を考慮 AWS Proton することはできません。これらの変更は表示されないためです AWS Proton。一貫性がない場合、結果は工ラーになります。

コンソールまたは を使用して AWS CLI 、環境テンプレートを登録および公開できます。

AWS マネジメントコンソール

コンソールで、新しい環境テンプレートを登録してパブリッシュします。

1. AWS Proton コンソールで [Environment templates (環境テンプレート)] を選択します。
2. [Create Environment template] (環境テンプレートの作成) を選択します。
3. [Create environment template (環境テンプレートを作成する)] ページの [Template options (テンプレートオプション)] セクションで利用可能な 2 つのテンプレートオプションのいずれかを選択します。

   - Create a template for provisioning new environments (新しい環境をプロビジョニング用のテンプレートを作成する)
   - 管理しようとするプロビジョニングされたインフラストラクチャを使用するテンプレートを作成します。
4. [Create a template for provisioning new environments (新しい環境のプロビジョニング用のテンプレートを作成する)] を選択した場合、[Template bundle source (テンプレートバンドルソース)] セクションで利用可能な 3 つのテンプレートバンドルソースオプションのいずれかを選択します。テンプレートの同期に関する要件と前提条件については、「テンプレートの同期設定」を参照してください。

   - Use one of our sample template bundles (サンプルテンプレートバンドルのいずれかを使用する)
   - Use your own template bundle (独自のテンプレートバンドルを使用する)
   - Sync templates from Git (Git からテンプレートを同期する)
5. テンプレートバンドルのパスを指定します。

   a. [Use one of our sample template bundles (サンプルテンプレートバンドルのいずれかを使用する)] を選択した場合:

      [Sample template bundle (サンプルテンプレートバンドル)] セクションでサンプルテンプレートバンドルを選択します。

   b. [Sync templates from Git (Git からテンプレートを同期する)] を選択した場合、[Source code (ソースコード)] セクションで以下の操作をします。

      i. テンプレート同期設定のリポジトリを選択します。

ii. 同期するリポジトリブランチの名前を入力します。

iii. (オプション) ディレクトリ名を入力して、テンプレートバンドルの検索を制限します。

c. それ以外の場合、[S3 bundle location (S3 バンドルの場所)] セクションでテンプレートバンドルのパスを指定します。

6. [Template details (テンプレートの詳細)] セクションで以下の操作をします。

a. テンプレート名を入力します。

b. (オプション) [Template display name] にテンプレート表示名を入力します。

c. (オプション) [Template description] に環境テンプレートの説明を入力します。

7. (オプション) [Encryption settings (暗号化設定)] セクションの [Customize encryption settings (advanced) (暗号化設定のカスタマイズ (アドバンスト))] のチェックボックスをオンにして、あなた自身の暗号化キーを提供します。

8. (オプション) [Tags (タグ)] セクションで [Add new tag (新しいタグを追加)] を選択し、キーと値を入力してカスタマーマネージドタグを作成します。

9. [Create Environment template (環境テンプレートの作成)] を選択します。

新しいページが開き、新しい環境テンプレートのステータスと詳細が表示されます。これらの詳細には、 AWS およびカスタマーマネージドタグのリストが含まれます。 は、 AWS Proton リソースの作成時に AWS マネージドタグ AWS Proton を自動的に生成します。詳細については、「AWS Proton リソースとタグ付け」を参照してください。

10. 新しい環境テンプレートのステータスは [Draft (ドラフト)] から始まります。その表示やアクセスができるのは、あなたと `proton:CreateEnvironment` 権限がある他のユーザーです。次の手順に従って、テンプレートを自分以外のユーザーが使用できるようにします。

11. [Template versions (テンプレートのバージョン)] セクションで、先ほど作成したテンプレートのマイナーバージョン (1.0) の左側にあるラジオボタンを選択します。別の方法としては、情報アラートで [Publish (パブリッシュ)] を選択して次のステップをスキップします。

12. [Template versions (テンプレートのバージョン)] セクションで [Publish (パブリッシュ)] を選択します。

13. テンプレートのステータスが [Published (パブリッシュ)] に変わります。最新バージョンのテンプレートなので、これは [Recommended (推奨)] バージョンです。

14. ナビゲーションペインで [Environment templates (環境テンプレート)] を選択して環境テンプレートと詳細の一覧を表示します。

コンソールを使用して、環境テンプレートの新しいメジャーバージョンとマイナーバージョンを登録します。

詳細については、「バージョン付きテンプレート」を参照してください。

1. AWS Proton コンソールで [Environment Templates (環境テンプレート)] を選択します。
2. 環境テンプレートのリストで、メジャーバージョンまたはマイナーバージョンを作成したい環境テンプレートの名前を選択します。
3. 環境テンプレートの詳細ビューの [Template version (テンプレートのバージョン)] セクションで [Create new version (新しいバージョンの作成)] を選択します。
4. [Create a new environment template version (新しい環境テンプレートバージョンを作成する)] ページの [Template bundle source (テンプレートバンドルソース)] セクションで利用可能な 2 つのテンプレートバンドルソースオプションのいずれかを選択します。
   - Use one of our sample template bundles (サンプルテンプレートバンドルのいずれかを使用する)
   - Use your own template bundle (独自のテンプレートバンドルを使用する)
5. 選択したテンプレートバンドルのパスを指定します。
   - [Use one of our sample template bundles (サンプルテンプレートバンドルのいずれかを使用する)] を選択したば場合、[Sample template bundle (サンプルテンプレートバンドル)] セクションでサンプルテンプレートバンドルを選択します。
   - [Use your own template bundle (独自のテンプレートバンドルを使用する)] を選択した場合、[S3 bundle location (S3 バンドルの場所)] セクションでテンプレートバンドルのパスを選択します。
6. [Template details (テンプレートの詳細)] セクションで以下の操作をします。
   a. (オプション) [Template display name] にテンプレート表示名を入力します。
   b. (オプション) [Template description] にサービステンプレートの説明を入力します。
7. [Template details (テンプレートの詳細)] セクションで、以下のオプションのいずれかを選択します。
   - マイナーバージョンを作成するには、[Check to create a new major version (チェックして新しいメジャーバージョンを作成)] チェックボックスをオフのままにしておきます。
   - メジャーバージョンを作成するには、[Check to create a new major version (チェックして新しいメジャーバージョンを作成)] チェックボックスをオンにします。

8. コンソールの手順を続けて、新しいマイナーバージョンまたはメジャーバージョンを作成し、[Create new version (新しいバージョンの作成)] を選択します。

AWS CLI

CLI を使用して、次の手順に示すように、新しい環境テンプレートを登録してパブリッシュします。

1. リージョン、名前、表示名 (オプション)、および説明 (オプション) を指定して、標準またはカスタマーマネージド環境テンプレートを作成します。

   a. 標準環境テンプレートを作成します。

      次のコマンドを実行してください。

```
$ aws proton create-environment-template \
    --name "simple-env" \
    --display-name "Fargate" \
    --description "VPC with public access"
```

      レスポンス:

```
{
    "environmentTemplate": {
        "arn": "arn:aws:proton:region-id:123456789012:environment-template/
simple-env",
        "createdAt": "2020-11-11T23:02:45.336000+00:00",
        "description": "VPC with public access",
        "displayName": "VPC",
        "lastModifiedAt": "2020-11-11T23:02:45.336000+00:00",
        "name": "simple-env"
    }
}
```

   b. provisioning である CUSTOMER_MANAGED パラメータを追加することでカスタマーマネージド環境テンプレートを作成します。

      次のコマンドを実行してください。

```
$ aws proton create-environment-template \
    --name "simple-env" \
```

```
    --display-name "Fargate" \
    --description "VPC with public access" \
    --provisioning "CUSTOMER_MANAGED"
```

レスポンス:

```
{
    "environmentTemplate": {
        "arn": "arn:aws:proton:region-id:123456789012:environment-template/
simple-env",
        "createdAt": "2020-11-11T23:02:45.336000+00:00",
        "description": "VPC with public access",
        "displayName": "VPC",
        "lastModifiedAt": "2020-11-11T23:02:45.336000+00:00",
        "name": "simple-env",
        "provisioning": "CUSTOMER_MANAGED"
    }
}
```

2. 環境テンプレートのメジャーバージョン 1 のマイナーバージョン 0 を作成します。

   このステップと残りのステップは、標準環境テンプレートとカスタマーマネージド環境テンプレートで同じです。

   環境テンプレートバンドルを含むバケットのテンプレート名、メジャーバージョン、S3 バケット名とキーを含めます。

   次のコマンドを実行してください。

```
$ aws proton create-environment-template-version \
    --template-name "simple-env" \
    --description "Version 1" \
    --source s3="{bucket=your_s3_bucket, key=your_s3_key}"
```

   レスポンス:

```
{
    "environmentTemplateVersion": {
        "arn": "arn:aws:proton:region-id:123456789012:environment-template/
simple-env:1.0",
        "createdAt": "2020-11-11T23:02:47.763000+00:00",
        "description": "Version 1",
```

```
        "lastModifiedAt": "2020-11-11T23:02:47.763000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "status": "REGISTRATION_IN_PROGRESS",
        "templateName": "simple-env"
    }
}
```

3. get コマンドを使用して登録ステータスを確認します。

   次のコマンドを実行してください。

```
$ aws proton get-environment-template-version \
    --template-name "simple-env" \
    --major-version "1" \
    --minor-version "0"
```

   レスポンス:

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment-template/
simple-env:1.0",
        "createdAt": "2020-11-11T23:02:47.763000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-11T23:02:47.763000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "recommendedMinorVersion": "0",
        "schema": "schema:\n  format:\n    openapi: \"3.0.0\"\n
 environment_input_type: \"MyEnvironmentInputType\"\n  types:\n
 MyEnvironmentInputType:\n      type: object\n      description: \"Input
 properties for my environment\"\n      properties:\n        my_sample_input:\n
          type: string\n          description: \"This is a sample input\"\n
      default: \"hello world\"\n        my_other_sample_input:\n          type:
 string\n          description: \"Another sample input\"\n      required:\n
   - my_other_sample_input\n",
        "status": "DRAFT",
        "statusMessage": "",
        "templateName": "simple-env"
    }
}
```

4. テンプレート名とメジャーバージョンとマイナーバージョンを指定して、環境テンプレートのメジャーバージョン 1 のマイナーバージョン 0 をパブリッシュします。このバージョンは Recommended バージョンです。

   次のコマンドを実行してください。

```
$ aws proton update-environment-template-version \
    --template-name "simple-env" \
    --major-version "1" \
    --minor-version "0" \
    --status "PUBLISHED"
```

   レスポンス:

```
{
    "environmentTemplateVersion": {
        "arn": "arn:aws:proton:region-id:123456789012:environment-template/
simple-env:1.0",
        "createdAt": "2020-11-11T23:02:47.763000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-11T23:02:54.610000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "recommendedMinorVersion": "0",
        "schema": "schema:\n  format:\n    openapi: \"3.0.0\"\n
 environment_input_type: \"MyEnvironmentInputType\"\n  types:\n
 MyEnvironmentInputType:\n      type: object\n      description: \"Input
 properties for my environment\"\n      properties:\n        my_sample_input:\n
          type: string\n          description: \"This is a sample input\"\n
      default: \"hello world\"\n        my_other_sample_input:\n          type:
 string\n          description: \"Another sample input\"\n      required:\n
   - my_other_sample_input\n",
        "status": "PUBLISHED",
        "statusMessage": "",
        "templateName": "simple-env"
    }
}
```

を使用して新しいテンプレートを作成したら AWS CLI、 AWS とカスタマーマネージドタグのリストを表示できます。 はマネージドタグ AWS Proton を自動的に生成 AWS します。また、

AWS CLIで、カスタマーマネージドタグの変更や作成もできます。詳細については、「AWS Proton リソースとタグ付け」を参照してください。

次のコマンドを実行してください。

```
$ aws proton list-tags-for-resource \
    --resource-arn "arn:aws:proton:region-id:123456789012:environment-
template/simple-env"
```

サービステンプレートを登録してパブリッシュする

サービステンプレートバージョンを作成する際に、互換性のある環境テンプレートの一覧を指定します。そうすれば、開発者はサービステンプレートを選択するときにサービスをデプロイする環境を選択できます。

サービステンプレートからサービスを作成する前や、サービステンプレートをパブリッシュする前は、リストにある互換性のある環境テンプレートから環境がデプロイされていることを確認してください。

互換性のない環境テンプレートから構築された環境にデプロイされたサービスは、新しいメジャーバージョンに更新できません。

サービステンプレートバージョンの互換性のある環境テンプレートを追加または削除するには、新しいメジャーバージョンを作成します。

コンソールまたは を使用して AWS CLI 、サービステンプレートを登録して公開できます。

AWS マネジメントコンソール

コンソールを使用して、新しいサービステンプレートを登録してパブリッシュします。

1. AWS Proton コンソールで [Service templates (サービステンプレート)] を選択します。
2. [Create service template (サービステンプレートの作成)] を選択します。
3. [Create service template (環境サービスを作成する)] ページの [Template bundle source (テンプレートバンドルソース)] セクションで利用可能なテンプレートオプションのいずれかを選択します。

   - Use your own template bundle (独自のテンプレートバンドルを使用する)
   - Sync templates from Git (Git からテンプレートを同期する)

4. テンプレートバンドルのパスを指定します。

   a. [Sync templates from Git (Git からテンプレートを同期する)] を選択した場合、[Source code repository (ソースコードリポジトリ)] セクションで以下の操作をします。

      i. テンプレート同期設定のリポジトリを選択します。

      ii. 同期するリポジトリブランチの名前を入力します。

      iii. (オプション) ディレクトリ名を入力して、テンプレートバンドルの検索を制限します。

   b. それ以外の場合、[S3 bundle location (S3 バンドルの場所)] セクションでテンプレートバンドルのパスを指定します。

5. [Template details (テンプレートの詳細)] セクションで以下の操作をします。

   a. テンプレート名を入力します。

   b. (オプション) [Template display name] にテンプレート表示名を入力します。

   c. (オプション) [Template description] にサービステンプレートの説明を入力します。

6. [Compatible environment templates (互換性のある環境テンプレート)] セクションで、リストから互換性のある環境テンプレートを選択します。

7. (オプション) [Encryption settings (暗号化設定)] セクションの [Customize encryption settings (advanced) (暗号化設定のカスタマイズ (アドバンスト))]] を選択し、あなた自身の暗号化キーを用意します。

8. (オプション) [パイプライン]セクション：

   あなたのサービステンプレートにサービスパイプライン定義を含めていない場合は、ページの下部にある [Pipeline - optional (パイプライン - オプション)] チェックボックスをオフにします。サービステンプレートの作成後にこれを変更することはできません。詳細については、「テンプレートバンドル」を参照してください。

9. (オプション)直接定義されたコンポーネントをサービスインスタンスにアタッチできるように、[サポート対象のコンポーネントソース] セクションの [コンポーネントソース] で、[直接定義] を選択します。

10. (オプション) [Tags (タグ)] セクションで [Add new tag (新しいタグを追加)] を選択し、キーと値を入力してカスタマーマネージドタグを作成します。

11. [Create service a template (サービステンプレートの作成)] を選択します。

新しいページが開き、新しいサービステンプレートのステータスと詳細が表示されます。これらの詳細には、 AWS およびカスタマーマネージドタグのリストが含まれます。 は、AWS Proton リソースの作成時に AWS マネージドタグ AWS Proton を自動的に生成します。詳細については、「AWS Proton リソースとタグ付け」を参照してください。

12. 新しいサービステンプレートステータスのステータスは [Draft (ドラフト)] から始まります。表示やアクセスできるのは、自分と自分以外の `proton:CreateService` アクセス許可があるユーザーです。次の手順に従って、テンプレートを自分以外のユーザーが使用できるようにします。
13. [Template versions (テンプレートのバージョン)] セクションで、先ほど作成したテンプレートのマイナーバージョン (1.0) の左側にあるラジオボタンを選択します。別の方法としては、情報アラートで [Publish (パブリッシュ)] を選択して次のステップをスキップします。
14. [Template versions (テンプレートのバージョン)] セクションで [Publish (パブリッシュ)] を選択します。
15. テンプレートのステータスが [Published (パブリッシュ)] に変わります。最新バージョンのテンプレートなので、これは [Recommended (推奨)] バージョンです。
16. ナビゲーションペインで [Service templates (サービステンプレート)] を選択してサービステンプレートと詳細の一覧を表示します。

コンソールを使用して、サービステンプレートの新しいメジャーバージョンとマイナーバージョンを登録します。

詳細については、「バージョン付きテンプレート」を参照してください。

1. AWS Proton コンソールで [Service Templates (サービステンプレート)] を選択します。
2. サービステンプレートのリストで、メジャーバージョンまたはマイナーバージョンを作成したいサービステンプレートの名前を選択します。
3. サービステンプレートの詳細ビューの [Template version (テンプレートのバージョン)] セクションで [Create new version] を選択します。
4. [サービステンプレートバージョンの新規作成] ページの [バンドルソース] セクションで、[あなた自身のテンプレートバンドルを使用] を選択します。
5. それ以外の場合、[S3 bundle location (S3 バンドルの場所)] セクションであなたのテンプレートバンドルまでのパスを選択します。
6. [Template details (テンプレートの詳細)] セクションで以下の操作をします。

    a. (オプション) [Template display name] にテンプレート表示名を入力します。

b. (オプション) [Template description] にサービステンプレートの説明を入力します。

7. [Template details (テンプレートの詳細)] セクションで、以下のオプションのいずれかを選択します。

   - マイナーバージョンを作成するには、[Check to create a new major version (チェックして新しいメジャーバージョンを作成)] チェックボックスをオフのままにしておきます。
   - メジャーバージョンを作成するには、[Check to create a new major version (チェックして新しいメジャーバージョンを作成)] チェックボックスをオンにします。

8. コンソールの手順を続けて、新しいマイナーバージョンまたはメジャーバージョンを作成し、[Create new version (新しいバージョンの作成)] を選択します。

AWS CLI

サービスパイプラインなしでサービスをデプロイするサービステンプレートを作成するには、パラメータと値 `--pipeline-provisioning "CUSTOMER_MANAGED"` を `create-service-template` コマンドに追加します。テンプレートバンドル 作成と サービステンプレートバンドルのスキーマ要件 の説明に従って、テンプレートバンドルを設定します。

> **Note**
>
> サービステンプレートの作成後に `pipelineProvisioning` を変更することはできません。

1. CLI を使用すると、次の手順に示すように、サービスパイプラインの有無にかかわらず、新しいサービステンプレートを登録してパブリッシュできます。

   a. CLI を使用してサービスパイプラインでサービステンプレートを作成します。

      名前、表示名 (オプション)、および説明 (オプション) を指定します。

      次のコマンドを実行してください。

```
$ aws proton create-service-template \
    --name "fargate-service" \
    --display-name "Fargate" \
    --description "Fargate-based Service"
```

レスポンス:

```
{
    "serviceTemplate": {
        "arn": "arn:aws:proton:region-id:123456789012:service-template/
fargate-service",
        "createdAt": "2020-11-11T23:02:55.551000+00:00",
        "description": "Fargate-based Service",
        "displayName": "Fargate",
        "lastModifiedAt": "2020-11-11T23:02:55.551000+00:00",
        "name": "fargate-service"
    }
}
```

b. サービスパイプラインを使用せずにサービステンプレートを作成します。

`--pipeline-provisioning` を追加します。

次のコマンドを実行します。

```
$ aws proton create-service-template \
    --name "fargate-service" \
    --display-name "Fargate" \
    --description "Fargate-based Service" \
    --pipeline-provisioning "CUSTOMER_MANAGED"
```

レスポンス:

```
{
    "serviceTemplate": {
        "arn": "arn:aws:proton:region-id:123456789012:service-template/
fargate-service",
        "createdAt": "2020-11-11T23:02:55.551000+00:00",
        "description": "Fargate-based Service",
        "displayName": "Fargate",
        "lastModifiedAt": "2020-11-11T23:02:55.551000+00:00",
        "name": "fargate-service",
        "pipelineProvisioning": "CUSTOMER_MANAGED"
    }
}
```

2. サービステンプレートのメジャーバージョン 1 のマイナーバージョン 0 を作成します。

   サービステンプレートバンドルを含むバケットのテンプレート名、互換性のある環境テンプレート、メジャーバージョン、S3 バケット名とキーを含めます。

   次のコマンドを実行してください。

```
$ aws proton create-service-template-version \
    --template-name "fargate-service" \
    --description "Version 1" \
    --source s3="{bucket=your_s3_bucket, key=your_s3_key}" \
    --compatible-environment-templates '[{"templateName":"simple-
env","majorVersion":"1"}]'
```

   レスポンス:

```
{
    "serviceTemplateMinorVersion": {
        "arn": "arn:aws:proton:region-id:123456789012:service-template/fargate-
service:1.0",
        "compatibleEnvironmentTemplates": [
            {
                "majorVersion": "1",
                "templateName": "simple-env"
            }
        ],
        "createdAt": "2020-11-11T23:02:57.912000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-11T23:02:57.912000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "status": "REGISTRATION_IN_PROGRESS",
        "templateName": "fargate-service"
    }
}
```

3. get コマンドを使用して登録ステータスを確認します。

   次のコマンドを実行します。

```
$ aws proton get-service-template-version \
```

```
    --template-name "fargate-service" \
    --major-version "1" \
    --minor-version "0"
```

レスポンス:

```
{
    "serviceTemplateMinorVersion": {
        "arn": "arn:aws:proton:us-east-1:123456789012:service-template/fargate-
service:1.0",
        "compatibleEnvironmentTemplates": [
            {
                "majorVersion": "1",
                "templateName": "simple-env"
            }
        ],
        "createdAt": "2020-11-11T23:02:57.912000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-11T23:02:57.912000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "schema": "schema:\n  format:\n    openapi: \"3.0.0\"\n
 pipeline_input_type: \"MyPipelineInputType\"\n  service_input_type:
 \"MyServiceInstanceInputType\"\n\n  types:\n    MyPipelineInputType:\n
    type: object\n      description: \"Pipeline input properties\"\n
 required:\n        - my_sample_pipeline_required_input\n      properties:\n
        my_sample_pipeline_optional_input:\n          type: string\n
   description: \"This is a sample input\"\n          default: \"hello world
\"\n        my_sample_pipeline_required_input:\n          type: string\n
     description: \"Another sample input\"\n\n    MyServiceInstanceInputType:
\n      type: object\n      description: \"Service instance input properties
\"\n      required:\n        - my_sample_service_instance_required_input\n
    properties:\n        my_sample_service_instance_optional_input:\n
   type: string\n          description: \"This is a sample input\"\n
 default: \"hello world\"\n        my_sample_service_instance_required_input:\n
         type: string\n          description: \"Another sample input\"",
        "status": "DRAFT",
        "statusMessage": "",
        "templateName": "fargate-service"
    }
}
```

4. update コマンドでステータスを "PUBLISHED" に変更して、サービステンプレートをパブリッシュします。

   次のコマンドを実行してください。

```
$ aws proton update-service-template-version \
    --template-name "fargate-service" \
    --description "Version 1" \
    --major-version "1" \
    --minor-version "0" \
    --status "PUBLISHED"
```

   レスポンス:

```
{
    "serviceTemplateVersion": {
        "arn": "arn:aws:proton:region-id:123456789012:service-template/fargate-
service:1.0",
        "compatibleEnvironmentTemplates": [
            {
                "majorVersion": "1",
                "templateName": "simple-env"
            }
        ],
        "createdAt": "2020-11-11T23:02:57.912000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-11T23:02:57.912000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "recommendedMinorVersion": "0",
        "schema": "schema:\n  format:\n    openapi: \"3.0.0\"\n
 pipeline_input_type: \"MyPipelineInputType\"\n  service_input_type:
 \"MyServiceInstanceInputType\"\n\n  types:\n    MyPipelineInputType:\n
    type: object\n      description: \"Pipeline input properties\"\n
 required:\n        - my_sample_pipeline_required_input\n      properties:\n
        my_sample_pipeline_optional_input:\n          type: string\n
 description: \"This is a sample input\"\n          default: \"hello pipeline
\"\n        my_sample_pipeline_required_input:\n          type: string\n
     description: \"Another sample input\"\n\n    MyServiceInstanceInputType:
\n      type: object\n      description: \"Service instance input properties
\"\n      required:\n        - my_sample_service_instance_required_input\n
    properties:\n        my_sample_service_instance_optional_input:\n
```

```
   type: string\n          description: \"This is a sample input\"\n
 default: \"hello world\"\n        my_sample_service_instance_required_input:\n
         type: string\n          description: \"Another sample input\"\n",
        "status": "PUBLISHED",
        "statusMessage": "",
        "templateName": "fargate-service"
    }
}
```

5. get コマンドを使用してサービステンプレートの詳細データを取得することで、 AWS Proton がバージョン 1.0 を公開していることを確認します。

   次のコマンドを実行してください。

```
$ aws proton get-service-template-version \
    --template-name "fargate-service" \
    --major-version "1" \
    --minor-version "0"
```

   レスポンス:

```
{
    "serviceTemplateMinorVersion": {
        "arn": "arn:aws:proton:us-east-1:123456789012:service-template/fargate-
service:1.0",
        "compatibleEnvironmentTemplates": [
            {
                "majorVersion": "1",
                "templateName": "simple-env"
            }
        ],
        "createdAt": "2020-11-11T23:02:57.912000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-11T23:03:04.767000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "schema": "schema:\n  format:\n    openapi: \"3.0.0\"\n
 pipeline_input_type: \"MyPipelineInputType\"\n  service_input_type:
 \"MyServiceInstanceInputType\"\n\n  types:\n    MyPipelineInputType:\n
    type: object\n      description: \"Pipeline input properties\"\n
 required:\n        - my_sample_pipeline_required_input\n      properties:\n
        my_sample_pipeline_optional_input:\n          type: string\n
   description: \"This is a sample input\"\n          default: \"hello world
```

```
\"\n        my_sample_pipeline_required_input:\n          type: string\n
     description: \"Another sample input\"\n\n    MyServiceInstanceInputType:
\n      type: object\n      description: \"Service instance input properties
\"\n      required:\n        - my_sample_service_instance_required_input\n
    properties:\n        my_sample_service_instance_optional_input:\n
   type: string\n          description: \"This is a sample input\"\n
 default: \"hello world\"\n        my_sample_service_instance_required_input:\n
         type: string\n          description: \"Another sample input\"",
        "status": "PUBLISHED",
        "statusMessage": "",
        "templateName": "fargate-service"
    }
}
```

テンプレートデータを表示する

AWS Proton コンソールと AWS CLIで、テンプレートと詳細を一覧表示し、詳細データを含む個々のテンプレートを表示できます。

カスタマーマネージド環境テンプレートデータには、provisioned 値がである CUSTOMER_MANAGED パラメータが含まれます。

サービステンプレートにサービスパイプラインが含まれていない場合、サービステンプレートデータに値が pipelineProvisioning である CUSTOMER_MANAGED パラメータが含まれます。

詳細については、「テンプレートを登録してパブリッシュする」を参照してください。

コンソールまたは を使用して AWS CLI 、テンプレートデータを一覧表示および表示できます。

AWS マネジメントコンソール

コンソールを使用してテンプレートを表示および一覧表示します。

1. テンプレートの一覧を表示するには、[(Environment or Service) templates ((環境またはサービス} テンプレート)] を選択します。
2. 詳細データを表示するには、テンプレートの名前を選択します。

   テンプレートの詳細データ、テンプレートのメジャーバージョンとマイナーバージョンのリスト、テンプレートバージョンとテンプレートタグを使用してデプロイされた AWS Proton リソースのリストを表示します。

推奨されるメジャーバージョンとマイナーバージョンには [Recommended (推奨)] のラベルが付きます。

AWS CLI

を使用してテンプレート AWS CLI を一覧表示および表示します。

次のコマンドを実行してください。

```
$ aws proton get-environment-template-version \
    --template-name "simple-env" \
    --major-version "1" \
    --minor-version "0"
```

レスポンス:

```
{
    "environmentTemplateVersion": {
        "arn": "arn:aws:proton:region-id:123456789012:environment-template/simple-
env:1.0",
        "createdAt": "2020-11-10T18:35:08.293000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-10T18:35:11.162000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "recommendedMinorVersion": "0",
        "schema": "schema:\n  format:\n    openapi: \"3.0.0\"\n
 environment_input_type: \"MyEnvironmentInputType\"\n  types:\n
 MyEnvironmentInputType:\n      type: object\n      description: \"Input properties
 for my environment\"\n      properties:\n        my_sample_input:\n
  type: string\n          description: \"This is a sample input\"\n
 default: \"hello world\"\n        my_other_sample_input:\n          type: string
\n          description: \"Another sample input\"\n      required:\n        -
 my_other_sample_input\n",
        "status": "DRAFT",
        "statusMessage": "",
        "templateName": "simple-env"
    }
}
```

次のコマンドを実行してください。

```
$ aws proton list-environment-templates
```

レスポンス:

```
{
    "templates": [
        {
            "arn": "arn:aws:proton:region-id:123456789012:environment-template/
simple-env-3",
            "createdAt": "2020-11-10T18:35:05.763000+00:00",
            "description": "VPC with Public Access",
            "displayName": "VPC",
            "lastModifiedAt": "2020-11-10T18:35:05.763000+00:00",
            "name": "simple-env-3",
            "recommendedVersion": "1.0"
        },
        {
            "arn": "arn:aws:proton:region-id:123456789012:environment-template/
simple-env-1",
            "createdAt": "2020-11-10T00:14:06.881000+00:00",
            "description": "Some SSM Parameters",
            "displayName": "simple-env-1",
            "lastModifiedAt": "2020-11-10T00:14:06.881000+00:00",
            "name": "simple-env-1",
            "recommendedVersion": "1.0"
        }
    ]
}
```

サービステンプレートのマイナーバージョンを表示します。

次のコマンドを実行してください。

```
$ aws proton get-service-template-version \
    --template-name "fargate-service" \
    --major-version "1" \
    --minor-version "0"
```

レスポンス:

```
{
```

```
    "serviceTemplateMinorVersion": {
        "arn": "arn:aws:proton:us-east-1:123456789012:service-template/fargate-
service:1.0",
        "compatibleEnvironmentTemplates": [
            {
                "majorVersion": "1",
                "templateName": "simple-env"
            }
        ],
        "createdAt": "2020-11-11T23:02:57.912000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-11T23:02:57.912000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "schema": "schema:\n  format:\n    openapi: \"3.0.0\"\n
 pipeline_input_type: \"MyPipelineInputType\"\n  service_input_type:
 \"MyServiceInstanceInputType\"\n\n  types:\n    MyPipelineInputType:\n
    type: object\n      description: \"Pipeline input properties\"\n
 required:\n        - my_sample_pipeline_required_input\n      properties:\n
       my_sample_pipeline_optional_input:\n          type: string\n
 description: \"This is a sample input\"\n          default: \"hello world\"\n
  my_sample_pipeline_required_input:\n          type: string\n          description:
 \"Another sample input\"\n\n    MyServiceInstanceInputType:\n      type: object
\n      description: \"Service instance input properties\"\n      required:\n
       - my_sample_service_instance_required_input\n      properties:\n
 my_sample_service_instance_optional_input:\n          type: string\n
 description: \"This is a sample input\"\n          default: \"hello world\"\n
     my_sample_service_instance_required_input:\n          type: string\n
 description: \"Another sample input\"",
        "status": "DRAFT",
        "statusMessage": "",
        "templateName": "fargate-service"
    }
}
```

次に示すコマンドとレスポンスの例では、サービスパイプラインなしでサービスを表示します。

次のコマンドを実行してください。

```
$ aws proton get-service-template \
    --name "simple-svc-template-cli"
```

レスポンス:

```
{
    "serviceTemplate": {
        "arn": "arn:aws:proton:region-id:123456789012:service-template/simple-svc-
template-cli",
        "createdAt": "2021-02-18T15:38:57.949000+00:00",
        "displayName": "simple-svc-template-cli",
        "lastModifiedAt": "2021-02-18T15:38:57.949000+00:00",
        "status": "DRAFT",
        "name": "simple-svc-template-cli",
        "pipelineProvisioning": "CUSTOMER_MANAGED"
    }
}
```

テンプレートを更新する

次の一覧の説明に従って、テンプレートを更新できます。

- コンソールまたは AWS CLIの使用時にテンプレートの `description` または `display name` を編集します。テンプレートの `name` を編集することはできません。
- コンソールまたは AWS CLIを使用するときに、テンプレートマイナーバージョンのステータスを更新します。ステータスの変更は `DRAFT` から `PUBLISHED` のみ可能です。
- AWS CLIの使用時に、テンプレートのマイナーバージョンまたはメジャーバージョンの表示名と説明を編集します。

AWS マネジメントコンソール

以下で説明する手順に従って、コンソールでテンプレートの説明と表示名を編集します。

テンプレートのリストで以下の操作をします。

1. AWS Proton コンソールで [(Environment or Service) Templates ((環境またはサービス) テンプレート)] を選択します。
2. テンプレートのリストで、説明または表示名を更新するテンプレートの左にあるラジオボタンを選択します。
3. [Actions (アクション)] を選択してから [Edit (編集)] を選択します。

4. [Edit (environment or service) template ((環境またはサービス) テンプレートの編集)] ページの [Template details (テンプレートの詳細)] セクションで、フォームに編集内容を入力して [Save changes (変更の保存)] を選択します。

コンソールを使用してテンプレートのマイナーバージョンのステータスを変更し、次の説明に従ってテンプレートをパブリッシュします。ステータスの変更は DRAFT から PUBLISHED のみ可能です。

テンプレート (環境またはサービステンプレート) の詳細ページ。

1. AWS Proton コンソールで[(Environment or Service) Templates ((環境またはサービス) テンプレート)] を選択します。
2. テンプレートのリストで、マイナーバージョンのステータスを [Draft (ドラフト)] から [Published (パブリッシュ)] に更新したいテンプレートの名前を選択します。
3. テンプレート (環境またはサービステンプレート) の詳細ページにある [Template versions (テンプレートのバージョン)] セクションで、パブリッシュしたいマイナーバージョンの左にあるラジオボタンを選択します。
4. [Template versions (テンプレートのバージョン)] セクションで [Publish (パブリッシュ)] を選択します。ステータスが [Draft (ドラフト)] から [Published (パブリッシュ)] に代わります。

AWS CLI

次のコマンドとレスポンスの例は、環境テンプレートの説明を編集する方法を示しています。

以下のコマンドを実行してください。

```
$ aws proton update-environment-template \
    --name "simple-env" \
    --description "A single VPC with public access"
```

レスポンス:

```
{
    "environmentTemplate": {
        "arn": "arn:aws:proton:region-id:123456789012:environment-template/simple-
env",
        "createdAt": "2020-11-28T22:02:10.651000+00:00",
        "description": "A single VPC with public access",
        "displayName": "simple-env",
```

```
        "lastModifiedAt": "2020-11-29T16:11:18.956000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "recommendedMinorVersion": "0",
        "schema": "schema:\n  format:\n    openapi: \"3.0.0\"\n
 environment_input_type: \"MyEnvironmentInputType\"\n  types:\n
 MyEnvironmentInputType:\n      type: object\n      description: \"Input properties
 for my environment\"\n      properties:\n        my_sample_input:\n
  type: string\n          description: \"This is a sample input\"\n
 default: \"hello world\"\n        my_other_sample_input:\n          type: string
\n          description: \"Another sample input\"\n      required:\n        -
 my_other_sample_input\n",
        "status": "PUBLISHED",
        "statusMessage": "",
        "templateName": "simple-env"
    }
}
```

を使用してサービステンプレート AWS CLI を更新することもできます。サービステンプレートのマイナーバージョンのステータスを更新する例については、ステップ 5 の「サービステンプレートを登録してパブリッシュする」を参照してください。

テンプレートを削除する

テンプレートはコンソールと AWS CLIを使用して削除できます。

そのバージョンにデプロイされた環境がない場合は、環境テンプレートのマイナーバージョンを削除できます。

そのバージョンにデプロイされたサービスインスタンスまたはパイプラインがない場合は、サービステンプレートのマイナーバージョンを削除できます。パイプラインは、サービスインスタンスとは異なるテンプレートバージョンにデプロイできます。たとえば、サービスインスタンスが 1.0 からバージョン 1.1 に更新され、パイプラインがバージョン 1.0 にデプロイされている場合、サービステンプレート 1.0 を削除することはできません。

AWS マネジメントコンソール

コンソールを使用して、テンプレート全体またはテンプレートの個々のマイナーバージョンとメジャーバージョンを削除できます。

コンソールを使用して、以下のようにテンプレートを削除します。

> **Note**
>
> コンソールを使用してテンプレートを削除する際には、
>
> - テンプレート全体を削除すると、テンプレートのメジャーバージョンとマイナーバージョンも削除されます。

テンプレート (環境またはサービステンプレート) のリスト。

1. AWS Proton コンソールで [(Environment or Service) Templates ((環境またはサービス) テンプレート)] を選択します。
2. テンプレートのリストで、削除したいテンプレートの左にあるラジオボタンを選択します。

   テンプレート全体を削除できるのは、そのバージョンにデプロイされた AWS Proton リソースがない場合のみです。
3. [Actions (アクション)] を選択してから [Delete (削除)] を選択してテンプレート全体を削除します。
4. モーダルから Delete アクションの確認を求めるプロンプトが表示されます。
5. 手順に従って操作し、[Yes, delete (はい、削除します)] を選択します。

テンプレート (環境またはサービステンプレート) の詳細ページ。

1. AWS Proton コンソールで [(Environment or Service) Templates ((環境またはサービス) テンプレート)] を選択します。
2. テンプレートのリストで、テンプレートの個々のメジャーバージョンまたはマイナーバージョンの全体を削除するか、または削除したいテンプレートの名前を選択します。
3. テンプレート全体を削除するには、以下のように操作します。

   テンプレート全体を削除できるのは、そのバージョンにデプロイされた AWS Proton リソースがない場合のみです。

   a. ページの右上にある [Deleet (削除)] を選択します。
   b. モーダルから Delete アクションの確認を求めるプロンプトが表示されます。
   c. 指示に従って操作し、[Yes, delete (はい、削除します)] を選択します。

4. テンプレートのメジャーバージョンまたはマイナーバージョンを削除するには

   テンプレートのマイナーバージョンは、そのバージョンにデプロイされた AWS Proton リソースがない場合にのみ削除できます。

   a. [Template versions (テンプレートのバージョン)] で、削除したいバージョンの左にあるラジオボタンを選択します。
   b. [Template versions (テンプレートのバージョン)] セクションで [Delete (削除)] を選択します。
   c. モーダルから Delete アクションの確認を求めるプロンプトが表示されます。
   d. 指示に従って操作し、[Yes, delete (はい、削除します)] を選択します。

AWS CLI

AWS CLI テンプレート削除オペレーションには、テンプレートの他のバージョンの削除は含まれません。を使用する場合は AWS CLI、次の条件でテンプレートを削除します。

- テンプレートのマイナーバージョンまたはメジャーバージョンが存在しない場合、テンプレート全体を削除します。
- 残っている最後のマイナーバージョンを削除するときにメジャーバージョンを削除します。
- そのバージョンにデプロイされた AWS Proton リソースがない場合は、テンプレートのマイナーバージョンを削除します。
- テンプレートの他のマイナーバージョンが存在しず、そのバージョンにデプロイされた AWS Proton リソースがない場合は、テンプレートの推奨マイナーバージョンを削除します。

次のコマンドとレスポンスの例は、 AWS CLI を使用してテンプレートを削除する方法を示しています。

次のコマンドを実行してください。

```
$ aws proton delete-environment-template-version \
    --template-name "simple-env" \
    --major-version "1" \
    --minor-version "0"
```

レスポンス:

```
{
```

```
    "environmentTemplateVersion": {
        "arn": "arn:aws:proton:region-id:123456789012:environment-template/simple-
env:1.0",
        "createdAt": "2020-11-11T23:02:47.763000+00:00",
        "description": "Version 1",
        "lastModifiedAt": "2020-11-11T23:02:54.610000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "status": "PUBLISHED",
        "statusMessage": "",
        "templateName": "simple-env"
    }
}
```

次のコマンドを実行してください。

```
$ aws proton delete-environment-template \
    --name "simple-env"
```

レスポンス:

```
{
    "environmentTemplate": {
        "arn": "arn:aws:proton:region-id:123456789012:environment-template/simple-
env",
        "createdAt": "2020-11-11T23:02:45.336000+00:00",
        "description": "VPC with Public Access",
        "displayName": "VPC",
        "lastModifiedAt": "2020-11-12T00:23:22.339000+00:00",
        "name": "simple-env",
        "recommendedVersion": "1.0"
    }
}
```

次のコマンドを実行してください。

```
$ aws proton delete-service-template-version \
    --template-name "fargate-service" \
    --major-version "1" \
    --minor-version "0"
```

レスポンス:

```
{
    "serviceTemplateVersion": {
        "arn": "arn:aws:proton:region-id:123456789012:service-template/fargate-
service:1.0",
        "compatibleEnvironmentTemplates": [{"majorVersion": "1", "templateName":
 "simple-env"}],
        "createdAt": "2020-11-28T22:07:05.798000+00:00",
        "lastModifiedAt": "2020-11-28T22:19:05.368000+00:00",
        "majorVersion": "1",
        "minorVersion": "0",
        "status": "PUBLISHED",
        "statusMessage": "",
        "templateName": "fargate-service"
    }
}
```

テンプレートの同期設定

定義した登録済み git リポジトリにあるテンプレートバンドルからの AWS Proton 同期を許可するようにテンプレートを設定する方法について説明します。あなたのリポジトリにコミットがプッシュされると、 AWS Proton は、あなたのリポジトリテンプレートバンドルに変更があるかどうかを調べます。テンプレートバンドルの変更を検出すると、そのテンプレートの新しいマイナーバージョンまたはメジャーバージョンがまだ存在しない場合、そのテンプレートが作成されます。 AWS Proton は現在、GitHub、GitHub Enterprise、BitBucket をサポートしています。

同期されたテンプレートバンドルにコミットをプッシュする

あなたのテンプレートのいずれかが追跡中のブランチにコミットをプッシュすると、 AWS Proton はあなたのリポジトリをクローンし、どのテンプレートで同期が必要とされているかを判断します。あなたのディレクトリ内のファイルをスキャンして、`{template-name}/{major-version}/` の規則に一致するディレクトリを検索します。

は、リポジトリとブランチに関連付けられているテンプレートとメジャーバージョン AWS Proton を決定すると、それらのすべてのテンプレートを並行して同期しようとします。

特定のテンプレートへの各同期中に、 は AWS Proton 最初にテンプレートディレクトリの内容が最後に成功した同期以降に変更されたかどうかを確認します。コンテンツが変更されなかった場合、重複バンドルの登録が AWS Proton スキップされます。そうすれば、テンプレートバンドルのコンテ

ンツが変更された場合に新しいテンプレートマイナーバージョンが作成されます。テンプレートバンドルの内容が変更された場合、バンドルは に登録されます AWS Proton。

テンプレートバンドルが登録されると、 は登録が完了するまで登録ステータスを AWS Proton モニタリングします。

特定のテンプレートのマイナーバージョンとメジャーバージョンに対して 1 つずつしか同期できません。同期の進行中にプッシュされた可能性のあるコミットはすべてバッチ処理されます。バッチされたコミットは、前回の同期試行の完了後に同期されます。

サービステンプレートを同期する

AWS Proton は、git リポジトリから環境テンプレートとサービステンプレートの両方を同期できます。あなたのサービステンプレートを同期するには、あなたのテンプレートバンドルの各メジャーバージョンディレクトリに `.template-registration.yaml` という名前のファイルを追加します。このファイルには、コミット後にサービステンプレートバージョンを作成するときに が AWS Proton 必要とする追加の詳細が含まれています。互換性のある環境とサポートされているコンポーネントソースです。

このファイルのフルパスは *`service-template-name`*`/`*`major-version`*`/.template-registration.yaml` です。詳細については、「the section called “サービステンプレートを同期する”」を参照してください。

テンプレート同期の設定に関する考慮事項

テンプレート同期設定の使用については、次の考慮事項を確認してください。

- リポジトリは 250 MB 以下にする必要があります。
- テンプレート同期を設定するには、まずリポジトリを AWS Protonにリンクします。詳細については、「the section called “レポジトリを作成する”」を参照してください。
- 同期されたテンプレートから新しいテンプレートバージョンが作成されると、DRAFT 状態になります。
- 次のいずれかに該当する場合、テンプレートの新しいマイナーバージョンが作成されます。
  - テンプレートバンドルの内容は、最後に同期されたテンプレートマイナーバージョンのものとは異なります。
  - 前回同期したテンプレートのマイナーバージョンが削除されました。
- 同期を一時停止することはできません。

- 新しいマイナーバージョンとメジャーバージョンの両方が自動的に同期されます。
- テンプレート同期設定では、新しい最上位レベルテンプレートを作成することはできません。
- テンプレート同期設定を使用して、複数のリポジトリから 1 つのテンプレートに同期することはできません。
- ブランチの代わりにタグを使用することはできません。
- サービステンプレートを作成する際には、互換性のある環境テンプレートを指定します。
- 環境テンプレートを作成し、それを同じコミット内であなたのサービステンプレートの互換環境として追加できます。
- 単一のテンプレートのメジャーバージョンへの同期は 1 つずつ実行されます。同期中に新しいコミットが検出されると、それらはアクティブ同期の終わりでバッチ処理され適用されます。異なるテンプレートのメジャーバージョンへの同期は並行して進行します。
- テンプレートの同期元のブランチを変更すると、まず古いブランチからの継続的な同期が完了します。次いで、新しいブランチから同期が始まります。
- テンプレートの同期元のリポジトリを変更すると、古いリポジトリからの継続的な同期が失敗したり、完了するまで実行される可能性があります。同期のどの段階にいるかによって異なります。

詳細については、「 AWS Proton サービス API リファレンス」を参照してください。

トピック

- テンプレート同期設定を作成する
- テンプレート同期設定の詳細を表示する
- テンプレート同期設定を編集する
- テンプレート同期設定を削除する

テンプレート同期設定を作成する

を使用してテンプレート同期設定を作成する方法について説明します AWS Proton。

テンプレート同期設定の前提条件を作成します。

- リポジトリを AWS Protonにリンクしました。
- テンプレートバンドルがリポジトリにあります。

リポジトリリンクは以下で構成されます。

- リポジトリにアクセスして通知をサブスクライブするアクセス AWS Proton 許可を付与する CodeConnections 接続。
- サービスリンクロール あなたのリポジトリをリンクすると、サービスリンクロールが作成されます。

最初のテンプレート同期設定を作成する前に、次のディレクトリレイアウトに示すように、テンプレートバンドルをリポジトリにプッシュします。

```
/templates/                                                 # subdirectory (optional)
/templates/my-env-template/                                 # template name
/templates/my-env-template/v1/                              # template version
/templates/my-env-template/v1/infrastructure/               # template bundle
/templates/my-env-template/v1/schema/
```

最初のテンプレート同期設定を作成した後、新しいバージョンの下 (たとえば、`/my-env-template/v2/` の下) に更新されたテンプレートバンドルを追加するコミットをプッシュすると、新しいテンプレートバージョンが自動的に作成されます。

```
/templates/                                                 # subdirectory (optional)
/templates/my-env-template/                                 # template name
/templates/my-env-template/v1/                              # template version
/templates/my-env-template/v1/infrastructure/               # template bundle
/templates/my-env-template/v1/schema/
/templates/my-env-template/v2/
/templates/my-env-template/v2/infrastructure/
/templates/my-env-template/v2/schema/
```

1 つ以上の同期設定済みテンプレートの新しいテンプレートバンドルバージョンを 1 つのコミットに含めることができます。 は、コミットに含まれていた新しいテンプレートバンドルバージョンごとに新しいテンプレートバージョン AWS Proton を作成します。

テンプレート同期設定を作成した後も、S3 バケットからテンプレートバンドルをアップロード AWS CLI することで、コンソールまたは でテンプレートの新しいバージョンを手動で作成できます。テンプレートの同期は、リポジトリから への 1 つの方向でのみ機能します AWS Proton。手動で作成したテンプレートのバージョンは同期されません。

テンプレート同期設定をセットアップすると、 はリポジトリへの変更を AWS Proton 待ちます。変更がプッシュされるたびに、テンプレートと同じ名前のディレクトリが検索されます。次に、そ

のディレクトリ内でメジャーバージョンのように見えるディレクトリを検索します。 は、対応するテンプレートメジャーバージョンにテンプレートバンドル AWS Proton を登録します。新しいバージョンは常に DRAFT 状態です。新しいバージョンは、 コンソールまたは を使用して公開できます AWS CLI。

たとえば、my-env-template というテンプレートがあってブランチ main の my-repo/templates から同期するように設定されているとすると、以下のようなレイアウトになります。

```
/code
/code/service.go
README.md
/templates/
/templates/my-env-template/
/templates/my-env-template/v1/
/templates/my-env-template/v1/infrastructure/
/templates/my-env-template/v1/schema/
/templates/my-env-template/v2/
/templates/my-env-template/v2/infrastructure/
/templates/my-env-template/v2/schema/
```

AWS Proton は の内容を /templates/my-env-template/v1/ にmy-env-template:1、 の内容を /templates/my-env-template/v2/に同期しますmy-env-template:2。これらのメジャーバージョンは、存在しない場合、作成されます。

AWS Proton は、テンプレート名に一致する最初のディレクトリを見つけました。テンプレート同期設定を作成または編集subdirectoryPathするときに を指定することで、ディレクトリ AWS Proton 検索を制限できます。たとえば、subdirectoryPath の /production-templates/ を指定できます。

テンプレート同期設定は、コンソールまたは CLI を使用して作成できます。

AWS マネジメントコンソール

テンプレートを使用して、テンプレートとテンプレート同期設定を作成します。

1. AWS Proton コンソールで [Environment templates (環境テンプレート)] を選択します。
2. [Create Environment template] (環境テンプレートの作成) を選択します。
3. [Create environment template (環境テンプレートを作成する)] ページの [Template options (テンプレートオプション)] セクションで [Create a template for provisioning new environments (新しい環境のプロビジョニング用のテンプレートを作成する)] を選択します。

4. [Template bundle source (テンプレートバンドルソース)] セクションで [Sync templates from Git (Git からテンプレートを同期する)] を選択します。
5. [Source code repository (ソースコードレポジトリ)] セクションで以下の操作をします。

   a. 「Repository」では、あなたのテンプレートバンドルがあるリンク先のリポジトリを選択します。

   b. [ブランチ] で、同期するリポジトリブランチを選択します。

   c. (オプション) テンプレートバンドルディレクトリには、あなたのテンプレートバンドルの検索を絞り込むためのディレクトリ名を入力します。
6. [Template details (テンプレートの詳細)] セクションで以下の操作をします。

   a. テンプレート名を入力します。

   b. (オプション) [Template display name] にテンプレート表示名を入力します。

   c. (オプション) [Template description] に環境テンプレートの説明を入力します。
7. (オプション) [Encryption settings (暗号化設定)] セクションの [Customize encryption settings (advanced) (暗号化設定のカスタマイズ (アドバンスト))] のチェックボックスをオンにします。
8. (オプション) [Tags (タグ)] セクションで [Add new tag (新しいタグを追加)] を選択し、キーと値を入力してカスタマーマネージドタグを作成します。
9. [Create Environment template (環境テンプレートの作成)] を選択します。

   新しいページが開き、新しい環境テンプレートのステータスと詳細が表示されます。これらの詳細には、 AWS マネージドタグとカスタマーマネージドタグのリストが含まれます。は、 AWS Proton リソースの作成時に AWS マネージドタグ AWS Proton を自動的に生成します。詳細については、「AWS Proton リソースとタグ付け」を参照してください。
10. テンプレートの詳細ページで [Sync (同期)] タブを使用してテンプレート同期設定の詳細データを表示します。
11. [Template versions (テンプレートのバージョン)] タブを選択して、テンプレートバージョンとステータスの詳細を表示します。
12. 新しい環境テンプレートのステータスは [Draft (ドラフト)] から始まります。その表示やアクセスができるのは、あなたと `proton:CreateEnvironment` 権限がある他のユーザーです。次の手順に従って、テンプレートを自分以外のユーザーが使用できるようにします。
13. [Template versions (テンプレートのバージョン)] セクションで、先ほど作成したテンプレートのマイナーバージョン (1.0) の左側にあるラジオボタンを選択します。別の方法としては、情報アラートで [Publish (発行)] を選択して次のステップをスキップします。

14. [Template versions (テンプレートのバージョン)] セクションで [Publish (パブリッシュ)] を選択します。
15. テンプレートのステータスが [Published (パブリッシュ済み)] に変わります。これはテンプレートの最新かつ推奨バージョンです。
16. ナビゲーションペインで [Environment templates (環境テンプレート)] を選択して環境テンプレートと詳細の一覧を表示します。

サービステンプレートとテンプレート同期設定を作成する手順も同様です。

AWS CLI

AWS CLIを使用して、テンプレートとテンプレート同期設定を作成します。

1. テンプレートを作成する この例では、環境テンプレートが作成されます。

   以下のコマンドを実行してください。

   ```
   $ aws proton create-environment-template \
       --name "env-template"
   ```

   レスポンスは次のとおりです。

   ```
   {
       "environmentTemplate": {
           "arn": "arn:aws:proton:us-east-1:123456789012:environment-template/env-template",
           "createdAt": "2021-11-07T23:32:43.045000+00:00",
           "displayName": "env-template",
           "lastModifiedAt": "2021-11-07T23:32:43.045000+00:00",
           "name": "env-template",
           "status": "DRAFT",
           "templateName": "env-template"
       }
   }
   ```

2. 以下を指定 AWS CLI して、 でテンプレート同期設定を作成します。

   - 同期先のテンプレート。テンプレート同期設定の作成後も、コンソールまたは AWS CLIを通してテンプレート同期設定から手動で新しいバージョンを作成できます。
   - テンプレート名。

- テンプレートタイプ。
- 希望する同期元のリンク先リポジトリ。
- リンクされたリポジトリプロバイダです。
- テンプレートバンドルが配置されているブランチ。
- (オプション) あなたのテンプレートバンドルがあるディレクトリパス。デフォルトでは、はテンプレート名に一致する最初のディレクトリ AWS Proton を探します。

以下のコマンドを実行してください。

```
$ aws proton create-template-sync-config \
    --template-name "env-template" \
    --template-type "ENVIRONMENT" \
    --repository-name "myrepos/templates" \
    --repository-provider "GITHUB" \
    --branch "main" \
    --subdirectory "env-template/"
```

レスポンスは次のとおりです。

```
{
    "templateSyncConfigDetails": {
        "branch": "main",
        "repositoryName": "myrepos/templates",
        "repositoryProvider": "GITHUB",
        "subdirectory": "templates",
        "templateName": "env-template",
        "templateType": "ENVIRONMENT"
    }
}
```

3. テンプレートバージョンを公開するには、「テンプレートを登録してパブリッシュする」を参照してください。

サービステンプレートを同期する

前の例では、環境テンプレートを同期する方法を示します。サービステンプレートも同様です。サービステンプレートを同期するには、あなたのテンプレートバンドルの各メジャーバージョンディレクトリに `.template-registration.yaml` という名前のファイルを追加します。このファイルに

は、コミット後にサービステンプレートバージョンを作成するときに が AWS Proton 必要とする追加の詳細が含まれています。 AWS Proton コンソールまたは API を使用してサービステンプレートバージョンを明示的に作成する場合、これらの詳細を入力として指定します。このファイルはテンプレート同期のためにこれらの入力を置き換えます。

```
./templates/                                                    # subdirectory (optional)
 /templates/my-svc-template/                                    # service template name
 /templates/my-svc-template/v1/                                 # service template version
 /templates/my-svc-template/v1/.template-registration.yaml      # service template version
 properties
 /templates/my-svc-template/v1/instance_infrastructure/         # template bundle
 /templates/my-svc-template/v1/schema/
```

`.template-registration.yaml` ファイルには次の詳細が含まれています。

- 互換性のある環境 [必須] — これらの環境テンプレートとメジャーバージョンに基づく環境は、このサービステンプレートバージョンに基づくサービスと互換性があります。
- サポート対象のコンポーネントソース [オプション] — これらのソースを使用するコンポーネントは、このサービステンプレートバージョンに基づくサービスと互換性があります。指定しないと、コンポーネントをこれらのサービスにアタッチできません。コンポーネントの詳細については、「コンポーネント」を参照してください。

ファイルの YAML 構文は次のとおりです。

```
compatible_environments:
  - env-templ-name:major-version
  - ...
supported_component_sources:
  - DIRECTLY_DEFINED
```

1 つ以上の環境テンプレートとメジャーバージョンの組み合わせを指定します。`supported_component_sources` の指定はオプションです。サポートされている値は `DIRECTLY_DEFINED` だけです。

Example.template-registration.yaml

この例では、サービステンプレートバージョンは `my-env-template` 環境テンプレートのメジャーバージョン 1 と 2 と互換性があります。また、`another-env-template` 環境テンプレートのメジャーバージョン 1 と 3 とも互換性があります。このファイルは

supported_component_sources を指定していないため、このサービステンプレートバージョンに基づくサービスにはコンポーネントをアタッチできません。

```
compatible_environments:
  - my-env-template:1
  - my-env-template:2
  - another-env-template:1
  - another-env-template:3
```

> **Note**
>
> 以前は、 は互換性のある環境を指定.compatible-envsするために別のファイル AWS Proton を定義していました。 AWS Proton は、下位互換性のためにそのファイルとその形式をサポートしています。このファイルは、拡張性がなく、コンポーネントなどの新しい機能をサポートできないため、これ以上の使用はお勧めしません。

テンプレート同期設定の詳細を表示する

コンソールまたは CLI を使用して、テンプレートの同期設定の詳細データを表示します。

AWS マネジメントコンソール

コンソールを使用して、テンプレートの同期設定の詳細を表示します。

1. ナビゲーションペインで、[(環境またはサービス) テンプレート] を選択します。
2. 詳細データを表示するには、テンプレート同期設定を作成したテンプレートの名前を選択します。
3. テンプレートの詳細ページで [Sync (同期)] タブを選択してテンプレート同期設定の詳細データを表示します。

AWS CLI

AWS CLI 同期されたテンプレートを表示するには、 を使用します。

以下のコマンドを実行してください。

```
$ aws proton get-template-sync-config \
    --template-name "svc-template" \
```

```
    --template-type "SERVICE"
```

レスポンスは次のとおりです。

```
{
    "templateSyncConfigDetails": {
        "branch": "main",
        "repositoryProvider": "GITHUB",
        "repositoryName": "myrepos/myrepo",
        "subdirectory": "svc-template",
        "templateName": "svc-template",
        "templateType": "SERVICE"
    }
}
```

を使用して AWS CLI 、テンプレートの同期ステータスを取得します。

`template-version` を使用する場合、テンプレートのメジャーバージョンを入力します。

以下のコマンドを実行してください。

```
$ aws proton get-template-sync-status \
    --template-name "env-template" \
    --template-type "ENVIRONMENT" \
    --template-version "1"
```

テンプレート同期設定を編集する

`template-name` と `template-type` を除き、テンプレート同期設定パラメータは編集できます。

コンソールまたは CLI を使用してテンプレート同期設定を編集する方法を説明します。

AWS マネジメントコンソール

コンソールでテンプレート同期設定ブランチを編集します。

テンプレートのリストで以下の操作をします。

1. AWS Proton コンソールで [(Environment or Service) Templates ((環境またはサービス) テンプレート)] を選択します。

2. テンプレートのリストで、編集したいテンプレート同期設定を含むテンプレートの名前を選択します。
3. テンプレートの詳細ページで [Template Sync (テンプレート同期)] タブを選択します。
4. [Template Sync details (テンプレート同期の詳細)] セクションで [Edit (編集)] を選択します。
5. [編集] ページの [ソースコードリポジトリ] セクションの [ブランチ] でブランチを選択し、[設定を保存] を選択します。

AWS CLI

次のコマンドとレスポンスの例は、CLI を使用してテンプレート同期設定 **branch** を編集する方法を示します。

以下のコマンドを実行してください。

```
$ aws proton update-template-sync-config \
    --template-name "env-template" \
    --template-type "ENVIRONMENT" \
    --repository-provider "GITHUB" \
    --repository-name "myrepos/templates" \
    --branch "fargate" \
    --subdirectory "env-template"
```

レスポンスは次のとおりです。

```
{
    "templateSyncConfigDetails": {
        "branch": "fargate",
        "repositoryProvider": "GITHUB",
        "repositoryName": "myrepos/myrepo",
        "subdirectory": "templates",
        "templateName": "env-template",
        "templateType": "ENVIRONMENT"
    }
}
```

同様に、 を使用して AWS CLI 同期されたサービステンプレートを更新できます。

テンプレート同期設定を削除する

コンソールまたは CLI を使用してテンプレート同期設定を削除します。

AWS マネジメントコンソール

コンソールを使用してテンプレート同期設定を削除します。

1. テンプレートの詳細ページで [Sync (同期)] タブを選択します。
2. [Sync details (同期の詳細)] セクションで [Disconnect (切断)] を選択します。

AWS CLI

次のコマンドとレスポンスの例は、 を使用して同期されたテンプレート設定 AWS CLI を削除する方法を示しています。

以下のコマンドを実行してください。

```
$ aws proton delete-template-sync-config \
    --template-name "env-template" \
    --template-type "ENVIRONMENT"
```

レスポンスは次のとおりです。

```
{
    "templateSyncConfig": {
        "templateName": "env-template",
        "templateType": "ENVIRONMENT"
    }
}
```

サービス同期設定

サービス同期を使用すると、Git を使用して AWS Proton サービスを設定およびデプロイできます。サービス同期を使用して、Git リポジトリで定義された設定を使用して、 AWS Proton サービスへの初期デプロイと更新を管理できます。Git では、バージョントラッキングやプルリクエストなどの機能で、あなたのサービスの設定、管理、デプロイができます。サービス同期は、 AWS Proton と Git を組み合わせて、 AWS Proton テンプレートを通じて定義および管理される標準化されたインフラストラクチャをプロビジョニングするのに役立ちます。Service sync を利用すれば、あなたの Git リ

ポジトリ内のサービス定義が管理され、ツールの切り替え回数を減らすことができます。Git のみを使用する場合と比較して、 テンプレートの標準化と での AWS Proton デプロイは、インフラストラクチャの管理に費やす時間を短縮するのに役立ちます。 AWS Proton また、 は、開発者チームとプラットフォームチームの両方により高い透明性と監査可能性を提供します。

AWS Proton OPS ファイル

proton-ops ファイルは、 がサービスインスタンスの更新に使用される仕様ファイル AWS Proton を見つける場所を定義します。また、サービスインスタンスを更新する順序と、あるインスタンスから別のインスタンスに変更をプロモートするタイミングも定義されます。

この proton-ops ファイルでは、あなたのリンク先のリポジトリにある仕様ファイルまたは複数の仕様ファイルで、サービスインスタンスを同期できます。これを行うには、次の例のように proton-ops ファイルに同期ブロックを定義します。

例 ./configuration/proton-ops.yaml:

```
sync:
  services:
      frontend-svc:
          alpha:
              branch: dev
              spec: ./frontend-svc/test/frontend-spec.yaml
          beta:
              branch: dev
              spec: ./frontend-svc/test/frontend-spec.yaml
          gamma:
              branch: pre-prod
              spec: ./frontend-svc/pre-prod/frontend-spec.yaml
          prod-one:
              branch: prod
              spec: ./frontend-svc/prod/frontend-spec-second.yaml
          prod-two:
              branch: prod
              spec: ./frontend-svc/prod/frontend-spec-second.yaml
          prod-three:
              branch: prod
              spec: ./frontend-svc/prod/frontend-spec-second.yaml
```

前の例では、frontend-svc はサービス名で、alpha、beta、gamma、prod-one、prod-two、prod-three はインスタンスです。

spec ファイルは、すべてのインスタンスでも、proton-ops ファイル内で定義されているインスタンスのサブセットでもかまいません。ただし、少なくとも、ブランチ内で定義されたインスタンスと同期元の仕様が必要です。インスタンスが特定のブランチと spec ファイルの場所で proton-ops ファイルに定義されていないと、Service Sync ではそれらのインスタンスの作成や更新はできません。

以下の例は、spec がどのようになるかを示します。proton-ops ファイルはこれらの spec ファイルから同期されることに注意してください。

例**./frontend-svc/test/frontend-spec.yaml**:

```
proton: "ServiceSpec"
instances:
- name: "alpha"
  environment: "frontend-env"
  spec:
    port: 80
    desired_count: 1
    task_size: "x-small"
    image: "public.ecr.aws/z9d2n7e1/nginx:1.21.0"
- name: "beta"
  environment: "frontend-env"
  spec:
    port: 80
    desired_count: 1
    task_size: "x-small"
    image: "public.ecr.aws/z9d2n7e1/nginx:1.21.0"
```

例**./frontend-svc/pre-prod/frontend-spec.yaml**:

```
proton: "ServiceSpec"
instances:
- name: "gamma"
  environment: "frontend-env"
  spec:
    port: 80
    desired_count: 1
    task_size: "x-small"
    image: "public.ecr.aws/z9d2n7e1/nginx:1.21.0"
```

例**./frontend-svc/prod/frontend-spec-second.yaml**:

```
proton: "ServiceSpec"
instances:
- name: "prod-one"
  environment: "frontend-env"
  spec:
    port: 80
    desired_count: 1
    task_size: "x-small"
    image: "public.ecr.aws/z9d2n7e1/nginx:1.21.0"
- name: "prod-two"
  environment: "frontend-env"
  spec:
    port: 80
    desired_count: 1
    task_size: "x-small"
    image: "public.ecr.aws/z9d2n7e1/nginx:1.21.0"
- name: "prod-three"
  environment: "frontend-env"
  spec:
    port: 80
    desired_count: 1
    task_size: "x-small"
    image: "public.ecr.aws/z9d2n7e1/nginx:1.21.0"
```

インスタンスが同期されず、同期を試みても引き続き問題が発生する場合は、GetServiceInstanceSyncStatus API を呼び出すと問題を解決できる場合があります。

> **Note**
>
> サービス同期を使用しているお客様は、引き続き AWS Proton 制限の対象となります。

ブロッカー

サービス同期を使用して AWS Proton サービスを同期することで、サービス仕様を更新し、Git リポジトリからサービスインスタンスを作成および更新できます。ただし、 AWS マネジメントコンソール または を使用してサービスまたはインスタンスを手動で更新する必要がある場合があります AWS CLI。

AWS Proton は、サービスインスタンスの更新やサービスインスタンスの削除など AWS CLI、 AWS マネジメントコンソール または を通じて行った手動の変更を上書きしないようにします。その場

合、手動による変更を検出すると Service Sync は無効になり、 AWS Proton はサービス同期ブロッカーを自動的に作成します。

サービスに関連するブロッカーをすべて取得するには、サービスに関連付けられている `serviceInstance` をそれぞれ順番に実行する必要があります。

- `serviceName` だけで `getServiceSyncBlockerSummary` API を呼び出します。
- `serviceName` と `serviceInstanceName` で `getServiceSyncBlockerSummary` API を呼び出します。

これにより、最新のブロッカーとそれに関連するステータスのリストが返されます。ACTIVE とマークされているブロッカーがある場合は、`blockerId` と `resolvedReason` でそれぞれに AND を指定して `UpdateServiceSyncBlocker` API を呼び出して解決する必要があります。

サービスインスタンスを手動で更新または作成する場合、 はサービスインスタンスにサービス同期ブロッカー AWS Proton を作成します。 は他のすべてのサービスインスタンスの同期 AWS Proton を続行しますが、ブロッカーが解決されるまでこのサービスインスタンスの同期を無効にします。サービスからサービスインスタンスを削除すると、 はサービスにサービス同期ブロッカー AWS Proton を作成します。これにより AWS Proton 、ブロッカーが解決されるまで、 はサービスインスタンスを同期できなくなります。

アクティブなブロッカーがすべて揃ったら、`blockerId` と `resolvedReason` でアクティブなブロッカーごとに、 `UpdateServiceSyncBlocker` API を呼び出して問題をあなたが解決する必要があります。

を使用して AWS マネジメントコンソール、 に移動 AWS Proton して Service Sync タブを選択すると、サービス同期が無効になっているかどうかを判断できます。サービスやサービスインスタンスがブロックされている場合は、[有効化] ボタンが表示されます。サービス同期を有効にするには、[有効化] を選択します。

トピック

- サービス同期設定を作成する
- サービス同期の設定の詳細が表示されます
- サービス同期設定を編集する
- サービス同期設定を削除する

サービス同期設定を作成する

サービス同期設定は、 コンソールまたは を使用して作成できます AWS CLI。

AWS マネジメントコンソール

1. [Choose a service template (サービステンプレートを選択する)] ページでテンプレートを選択して [Configure (設定する)] を選択します。
2. [Configure service (サービスを構成する)] ページで [Service details (サービス詳細)] セクションの [Service name] に新しいサービス名を入力します。
3. (オプション) サービスの説明を入力します。
4. アプリケーションソースコードリポジトリセクションで、リンクされた Git リポジトリを選択して、既にリンクしているリポジトリを選択します AWS Proton。リンクされたリポジトリがまだない場合は、[別の Git リポジトリをリンク] を選択し、[リポジトリへのリンクを作成する] の指示に従ってください。
5. [Repository (リポジトリ)] であなたのソースコードが保存されているリポジトリの名前を選択します。
6. [Branch (ブランチ)] であなたのソースコードが保存されているリポジトリブランチの名前を選択します。
7. (オプション) [Tags (タグ)] セクションで [Add new tag (新しいタグを追加)] を選択し、キーと値を入力してカスタマーマネージドタグを作成します。
8. [Next (次へ)] をクリックします。
9. [サービスインスタンスの設定] ページの [サービス定義ソース] セクションで、[Git からサービスを同期] を選択します。
10. [サービス定義ファイル] セクションで、あなたの `proton-ops` ファイルを作成する場合は AWS Proton 、[AWS Proton にファイルを作成させたい] を選択します。このオプションを使用すると、 は指定した場所に `spec`と `proton-ops` ファイル AWS Proton を作成します。あなた自身の OPS ファイルを作成するには、[独自のファイルを提供しています] を選択します。
11. サービス定義リポジトリセクションで、リンクされた Git リポジトリを選択して、既にリンクしているリポジトリを選択します AWS Proton。
12. [Repository name (リポジトリ名)] でソースコードが含まれているリポジトリの名前を選択します。

13. **proton-ops** ファイルブランチの場合、 が OPS と仕様ファイルを配置 AWS Proton するリストからブランチの名前を選択します。
14. [サービスインスタンス] セクションでは、proton-ops ファイル内の値に基づいて各フィールドが自動的に入力されます。
15. [Next (次へ)] を選択して入力を見直します。
16. [作成] を選択します。

AWS CLI

を使用してサービス同期設定を作成する AWS CLI

- 以下のコマンドを実行してください。

```
$ aws proton create-service-sync-config \
    --resource "service-arn" \
    --repository-provider "GITHUB" \
    --repository "example/proton-sync-service" \
    --ops-file-branch "main" \
    --proton-ops-file "./configuration/custom-proton-ops.yaml" (optional)
```

レスポンスは次のとおりです。

```
{
    "serviceSyncConfig": {
        "branch": "main",
        "filePath": "./configuration/custom-proton-ops.yaml",
        "repositoryName": "example/proton-sync-service",
        "repositoryProvider": "GITHUB",
        "serviceName": "service name"
    }
}
```

サービス同期の設定の詳細が表示されます

サービス同期の設定詳細データは、コンソールまたは AWS CLIの表示で確認できます。

AWS マネジメントコンソール

コンソールを使用して、サービス同期の設定の詳細を表示します。

1. ナビゲーションペインで [Services (サービス)] を選択します。
2. 詳細データを表示するには、サービス同期設定を作成したサービスの名前を選択します。
3. サービスの詳細ページで [サービス同期] タブを選択すると、サービス同期の設定詳細データが表示されます。

AWS CLI

AWS CLI を使用して、同期されたサービスを取得します。

以下のコマンドを実行してください。

```
$ aws proton get-service-sync-config \
    --service-name "service name"
```

レスポンスは次のとおりです。

```
{
    "serviceSyncConfig": {
        "branch": "main",
        "filePath": "./configuration/custom-proton-ops.yaml",
        "repositoryName": "example/proton-sync-service",
        "repositoryProvider": "GITHUB",
        "serviceName": "service name"
    }
}
```

を使用して AWS CLI 、サービス同期ステータスを取得します。

以下のコマンドを実行してください。

```
$ aws proton get-service-sync-status \
    --service-name "service name"
```

サービス同期設定を編集する

サービス同期設定は、 コンソールまたは を使用して編集できます AWS CLI。

AWS マネジメントコンソール

コンソールでテンプレート同期設定を編集します。

1. ナビゲーションペインで [Services (サービス)] を選択します。
2. 詳細データを表示するには、サービス同期設定を作成したサービスの名前を選択します。
3. サービス詳細ページで、[サービス同期タブ] を選択します。
4. [Services Sync] セクションで、[Edit] を選択します。
5. [Edit] ページで、編集する情報を更新し、[Save] を選択します。

AWS CLI

次のコマンドとレスポンスの例は、 AWS CLIを使用してテンプレート同期設定を編集する方法を示します。

以下のコマンドを実行してください。

```
$ aws proton update-service-sync-config \
    --service-name "service name" \
    --repository-provider "GITHUB" \
    --repository "example/proton-sync-service" \
    --ops-file-branch "main" \
    --ops-file "./configuration/custom-proton-ops.yaml"
```

レスポンスは次のとおりです。

```
{
    "serviceSyncConfig": {
        "branch": "main",
        "filePath": "./configuration/custom-proton-ops.yaml",
        "repositoryName": "example/proton-sync-service",
        "repositoryProvider": "GITHUB",
        "serviceName": "service name"
    }
}
```

サービス同期設定を削除する

サービス同期設定は、 コンソールまたは を使用して削除できます AWS CLI。

AWS マネジメントコンソール

コンソールでサービス同期設定を削除します。

1. サービスの詳細ページで、[Service Sync]タブを選択します。
2. [サービス同期の詳細] セクションで、[Disconnect] を選択してあなたのリポジトリを切断します。あなたのリポジトリが切断されると、そのリポジトリからのサービスは同期されなくなります。

AWS CLI

次のコマンドとレスポンスの例は、 を使用してサービス同期設定 AWS CLI を削除する方法を示しています。

以下のコマンドを実行してください。

```
$ aws proton delete-service-sync-config \
    --service-name "service name"
```

レスポンスは次のとおりです。

```
{
    "serviceSyncConfig": {
        "branch": "main",
        "filePath": "./configuration/custom-proton-ops.yaml",
        "repositoryName": "example/proton-sync-service",
        "repositoryProvider": "GITHUB",
        "serviceName": "service name"
    }
}
```

> **Note**
>
> Service Sync ではサービスインスタンスは削除されません。設定のみを削除します。

AWS Proton 環境

環境は AWS Proton、サービスがデプロイされる AWS Proton 一連の共有リソースとポリシーを表します。 AWS Proton サービスインスタンス間で共有することが予想されるリソースを含めることができます。これらのリソースとしては、VPC、クラスタ、共有ロードバランサーまたは API ゲートウェイなどのリソースあります。サービスをデプロイする前に、 AWS Proton 環境を作成する必要があります。

このセクションでは、サービスの作成、表示、更新、および削除のオペレーションをしながらサービスを管理する方法について説明します。>追加情報については、「 AWS Proton サービス API リファレンス」を参照してください。

トピック

- IAM ロール
- 環境を作成する
- 環境データを表示する
- 環境を更新する
- 環境を削除する
- 環境アカウント接続
- カスタマーマネージド環境
- CodeBuild プロビジョニングロールの作成

IAM ロール

では AWS Proton、所有および管理している AWS リソースの IAM ロールと AWS KMS キーを指定します。その後、これらは開発者が所有し、管理するリソースに適用され、そこで使用されます。開発者チームの AWS Proton API へのアクセスを制御する IAM ロールを作成します。

AWS Proton サービスロール

新しい環境を作成するときは、関連する IAM サービスロールを指定します。ロールには、環境テンプレートとサービステンプレートの両方で定義されるプロビジョニング済みのインフラストラクチャのすべてを更新するときに必要なすべての権限があります。ロールの例については、AWS Proton を使用してプロビジョニングするための サービスロール CloudFormation を参照してください。環境ア

カウント接続および環境アカウントを使用する場合は、選択した環境アカウントにロールを作成します。詳細については、「1 つのアカウントに環境を作成し、別のアカウントでプロビジョニングします。」および「環境アカウント接続」を参照してください。

このサービスロールの提供方法と、誰がロールを引き受けるかは、あなたの環境のプロビジョニング方法によって異なります。

- AWSマネージドプロビジョニング – 環境の作成中に直接 AWS Proton、またはアカウント接続を介して間接的に にロールを提供します。 は、環境とサービスインフラストラクチャをプロビジョニングするために、関連するアカウントのロール AWS Proton を引き受けます。
- セルフマネージドプロビジョニング — あなたには、プルリクエスト (PR) でプロビジョニングアクションがトリガーされたときに、適切な認証情報を使用して適切なロールが引き受けられるようにプロビジョニング自動化を設定していただきます。ロールを引き受ける GitHub アクションの例については、GitHub Actions の「認証情報の設定」アクションドキュメントの「ロールの引き受け」を参照してください。 AWS GitHub

プロビジョニング方法の詳細については、「the section called “プロビジョニングの方法”」を参照してください。

環境を作成する

AWS Proton 環境の作成について説明します。

AWS Proton 環境は、次の 2 つの方法のいずれかで作成できます。

- 標準環境テンプレートを使用して、標準環境を作成、管理、プロビジョニングします。 は、環境のインフラストラクチャを AWS Proton プロビジョニングします。
- カスタマーマネージド環境テンプレートを使用して、カスタマーマネージドインフラストラクチャ AWS Proton に接続します。の外部で独自の共有リソースをプロビジョニングし AWS Proton、AWS Proton が使用できるプロビジョニング出力を提供します。

環境の作成方法としては、いくつかのプロビジョニング方法の中から 1 つを選択できます。

- AWS マネージドプロビジョニング – 1 つのアカウントで環境を作成、管理、プロビジョニングします。 は環境を AWS Proton プロビジョニングします。

  この方法でサポートされるのは CloudFormation Infrastructure Code (IaC) テンプレートのみです。

- AWS 別のアカウントへの マネージドプロビジョニング – 1 つの管理アカウントで、環境アカウント接続を使用して別のアカウントでプロビジョニングされた環境を作成および管理します。 は、別のアカウントの環境を AWS Proton プロビジョニングします。詳細については、「1 つのアカウントに環境を作成し、別のアカウントでプロビジョニングします。」および「環境アカウント接続」を参照してください。

  この方法でサポートされるのは CloudFormation IaC テンプレートのみです。

- セルフマネージドプロビジョニング – 独自のプロビジョニングインフラストラクチャを持つリンクされたリポジトリにプロビジョニングプルリクエスト AWS Proton を送信します。

  この方法でサポートされるのは Terraform IaC テンプレートのみです。

- CodeBuild プロビジョニング – 指定したシェルコマンドを実行する AWS CodeBuild ために AWS Proton を使用します。コマンドは、 AWS Proton が提供する入力を読み取ることができ、インフラストラクチャのプロビジョニングまたはプロビジョニング解除と出力値の生成を担当します。この方法のテンプレートバンドルには、マニフェストファイル内のあなたのコマンドと、これらのコマンドで必要になるプログラム、スクリプト、またはその他のファイルが含まれます。

  CodeBuild プロビジョニングを使用する例として、 を使用して AWS リソース AWS Cloud Development Kit (AWS CDK) をプロビジョニングするコードと、CDK をインストールして CDK コードを実行するマニフェストを含めることができます。

  詳細については、「the section called “CodeBuild バンドル”」を参照してください。

  > **Note**
  >
  > CodeBuild プロビジョニングは環境とサービスで使用できます。現時点では、この方法でコンポーネントをプロビジョニングすることはできません。

AWS マネージドプロビジョニング (同じアカウントと別のアカウントの両方) AWS Proton では、 はリソースをプロビジョニングするための直接呼び出しを行います。

セルフマネージドプロビジョニングを使用すると、 はプルリクエスト AWS Proton を行い、IaC エンジンがリソースのプロビジョニングに使用するコンパイル済み IaC ファイルを提供します。

詳細については、「the section called “プロビジョニングの方法”」、「the section called “テンプレートバンドル”」、および「the section called “環境スキーマ要件”」を参照してください。

トピック

- 同じアカウント内で標準環境を作成してプロビジョニングする
- 1 つのアカウントに環境を作成し、別のアカウントでプロビジョニングします。
- セルフマネージドプロビジョニングで環境を作成し、プロビジョニングします。

同じアカウント内で標準環境を作成してプロビジョニングする

コンソールまたは AWS CLI を使用して、1 つのアカウントで環境を作成してプロビジョニングします。プロビジョニングは によって管理されます AWS。

AWS マネジメントコンソール

コンソールを使用して、単一のアカウント内で環境を作成してプロビジョニングします。

1. AWS Proton コンソールで、[環境] を選択します。
2. [Create environment (環境の作成)] を選択します。
3. [Choose an environment template (環境テンプレートを選択する)] ページでテンプレートを選択して [Configure (設定)] を選択します。
4. [Configure environment (環境を設定する)] ページの [Provisioning (プロビジョニング)] セクションで [AWS マネージドプロビジョニング] を選択します。
5. [Deployment account (デプロイアカウント)] セクションで、[この AWS アカウント] を選択します。
6. [Configure environment (環境の設定)] ページの [Environment settings (環境設定)] セクションで [Environment name] に環境名を入力します。
7. (オプション) この環境についての説明を入力します。
8. [Environment roles (環境ロール)] セクションで、AWS Proton サービスロールの設定 の一部としてあなたが作成した AWS Proton サービスロールを選択します。
9. (オプション) [コンポーネントロール] セクションで、直接定義したコンポーネントを環境内で実行できるようにするサービスロールを選択し、プロビジョニングできるリソースの範囲を絞り込みます。詳細については、「コンポーネント」を参照してください。
10. (オプション) [Tags (タグ)] セクションで [Add new tag (新しいタグを追加)] を選択し、キーと値を入力してカスタマーマネージドタグを作成します。
11. [Next (次へ)] を選択します。
12. [Configure environment custom settings (環境カスタム設定の構成)] ページでは、`required` パラメータの値を入力してください。`optional` パラメータの値を入力するか、表示されるデフォルト値を使用します。

13. [Next (次へ)] を選択して入力を見直します。
14. [Create (作成)] を選択します。

    環境の詳細とステータス、ならびに環境に関する AWS マネージドタグとカスタマーマネージドタグ。
15. ナビゲーションペインで [Environments (環境)] を選択します。

    新しいページには、ステータスやその他の環境の詳細とともに環境のリストが表示されます。

AWS CLI

を使用して AWS CLI 、1 つのアカウントで環境を作成してプロビジョニングします。

環境を作成するには、AWS Proton サービスロール ARN、あなたの仕様ファイルまでのパス、環境名、環境テンプレート ARN、メジャーバージョンとマイナーバージョン、説明 (オプション) を指定します。

次の例は、環境テンプレートスキーマファイルで定義された 2 つの入力値を指定する YAML フォーマット済み仕様ファイルです。`get-environment-template-minor-version` コマンドを実行して、環境テンプレートスキーマを表示できます。

```
proton: EnvironmentSpec
spec:
  my_sample_input: "the first"
  my_other_sample_input: "the second"
```

次のコマンドを実行して、環境を作成します。

```
$ aws proton create-environment \
    --name "MySimpleEnv" \
    --template-name simple-env \
    --template-major-version 1 \
    --proton-service-role-arn "arn:aws:iam::123456789012:role/AWS ProtonServiceRole"
 \
    --spec "file://env-spec.yaml"
```

レスポンス:

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv",
        "createdAt": "2020-11-11T23:03:05.405000+00:00",
        "deploymentStatus": "IN_PROGRESS",
        "lastDeploymentAttemptedAt": "2020-11-11T23:03:05.405000+00:00",
        "name": "MySimpleEnv",
        "protonServiceRoleArn": "arn:aws:iam::123456789012:role/ProtonServiceRole",
        "templateName": "simple-env"
    }
}
```

新しい環境を作成したら、次の コマンド例に示すように、 AWS とカスタマーマネージドタグのリストを表示できます。 は AWS マネージドタグ AWS Proton を自動的に生成します。また、AWS CLIで、カスタマーマネージドタグの変更や作成もできます。詳細については、「AWS Proton リソースとタグ付け」を参照してください。

コマンド:

```
$ aws proton list-tags-for-resource \
    --resource-arn "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv"
```

1 つのアカウントに環境を作成し、別のアカウントでプロビジョニングします。

コンソールまたは AWS CLI を使用して、別のアカウントの環境インフラストラクチャをプロビジョニングする標準環境を管理アカウントに作成します。プロビジョニングは AWSが管理します。

コンソールまたは CLI を使用する前に、以下のステップを完了します。

1. 管理アカウントと環境アカウントの AWS アカウント IDs を特定し、後で使用するためにコピーします。
2. 環境アカウントで、環境が作成する最小限のアクセス許可を持つ AWS Proton サービスロールを作成します。詳細については、「AWS Proton を使用してプロビジョニングするための サービスロール CloudFormation」を参照してください。

AWS マネジメントコンソール

コンソールを使用して、1 つのアカウントに環境を作成し、別のアカウントでプロビジョニングします。

1. 環境アカウント内で環境アカウント接続を作成し、それを使用して管理アカウントへの接続リクエストを送信します。

   a. AWS Proton コンソールでナビゲーションペインにある [Environment account connections (環境アカウント接続)] を選択します。

   b. [Environment account connections (環境アカウント接続)] ページで[Request to connect (接続のリクエスト)] を選択します。

   > **Note**
   >
   > [Environment account connection (環境アカウント接続ページ)] の見出しに挙がっているアカウント ID が、事前に識別されたあなたの環境アカウント ID と一致することを確認してください。

   c. [Request to connect (接続のリクエスト)] ページでの [Environment role (環境ロール)] セクションで [Existing service role (既存のサービスロール)] および環境用に作成したサービスロールの名前を選択します。

   d. 「管理アカウントへの接続」セクションに、環境の管理アカウント ID と環境名を入力します AWS Proton 。後で使用できるように名前をコピーします。

   e. ページの右下にある [Request to connect (接続のリクエスト)] を選択します。

   f. リクエストは [Environment connections sent to a management account (管理アカウントに送信される環境接続)] テーブルに [Pending] (保留中) として表示され、モーダルに管理アカウントからのリクエストを受け入れる方法が示されます。

2. 管理アカウント内で、環境アカウントからの接続リクエストを受け入れ諾ます。

   a. 管理アカウントにログインし、 AWS Proton コンソールで環境アカウント接続を選択します。

   b. [Environment account connections (環境アカウント接続の)] ページの [Environment account connection requests (環境アカウント接続リクエスト)] テーブルで、事前に識別された環境アカウント ID と一致する環境アカウント ID を持つ環境アカウント接続を選択します。

> **Note**
>
> [Environment account connection (環境アカウント接続ページ)] の見出しに挙がっているアカウント ID が、事前に識別されたあなたの管理アカウント ID と一致することを確認してください。

c. [Accept (承諾)] を選択します。ステータスが [Pending (保留中)] から [Connected (接続済み)] に変わります。

3. 管理アカウント内で環境を作成します。

a. ナビゲーションペインで [Environment templates (環境テンプレート)] を選択します。

b. [Environment templates (環境テンプレート)] ページで [Create environment template (環境テンプレートを作成する)] を選択します。

c. [Choose an environment template (環境テンプレートを選択する)] ページで環境テンプレートを選択します。

d. [Configure environment (環境を設定する)] ページの [Provisioning (プロビジョニング)] セクションで [AWS マネージドプロビジョニング] を選択します。

e. デプロイアカウントセクションで、別の AWS アカウントを選択します。

f. [Environment details (環境の詳細)] セクションで、[環境アカウント接続] と [環境名] を選択します。

g. [Next (次へ)] をクリックします。

h. フォームに必要な値を入力し、[Review and Create (確認と作成)] ページが表示されるまで [Next (次へ)] を選択します。

i. 内容を見直して [Create environment (環境を作成する)] を選択します。

AWS CLI

を使用して、あるアカウントに環境 AWS CLI を作成し、別のアカウントにプロビジョニングします。

環境アカウント内で環境アカウント接続を作成し、以下のコマンドを実行して接続をリクエストします。

```
$ aws proton create-environment-account-connection \
    --environment-name "simple-env-connected" \
```

```
    --role-arn "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role" \
    --management-account-id "111111111111"
```

レスポンス:

```
{
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:region-id:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-connected",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:13:50.847000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role",
        "status": "PENDING"
    }
}
```

管理アカウント内で、以下のコマンドを実行して環境アカウント接続リクエストを受け入れます。

```
$ aws proton accept-environment-account-connection \
    --id "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

レスポンス:

```
{
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:region-id:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-connected",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:15:33.486000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role",
```

```
        "status": "CONNECTED"
    }
}
```

次のコマンドを実行してあなたの環境アカウント接続を表示します。

```
$ aws proton get-environment-account-connection \
    --id "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

レスポンス:

```
{
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:region-id:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-connected",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:15:33.486000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role",
        "status": "CONNECTED"
    }
}
```

管理アカウントで、次のコマンドを実行して環境を作成します。

```
$ aws proton create-environment \
    --name "simple-env-connected" \
    --template-name simple-env-template \
    --template-major-version "1" \
    --template-minor-version "1" \
    --spec "file://simple-env-template/specs/original.yaml" \
    --environment-account-connection-id "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

レスポンス:

```
{
    "environment": {
```

```
        "arn": "arn:aws:proton:region-id:111111111111:environment/simple-env-
connected",
        "createdAt": "2021-04-28T23:02:57.944000+00:00",
        "deploymentStatus": "IN_PROGRESS",
        "environmentAccountConnectionId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "lastDeploymentAttemptedAt": "2021-04-28T23:02:57.944000+00:00",
        "name": "simple-env-connected",
        "templateName": "simple-env-template"
    }
}
```

セルフマネージドプロビジョニングで環境を作成し、プロビジョニングします。

セルフマネージドプロビジョニングを使用する場合、 は独自のプロビジョニングインフラストラクチャを持つリンクされたリポジトリにプロビジョニングプルリクエスト AWS Proton を送信します。プルリクエストは独自のワークフローを開始し、 AWS サービスを呼び出してインフラストラクチャをプロビジョニングします。

セルフマネージドプロビジョニングに関する考察：

- 環境を作成する前に、セルフマネージドプロビジョニング用のリポジトリリソースディレクトリを設定します。詳細については、「AWS Proton コードファイルとしての インフラストラクチャ」を参照してください。
- 環境を作成すると、 はインフラストラクチャプロビジョニングのステータスに関する非同期通知を受信するのを AWS Proton 待ちます。プロビジョニングコードは、 AWS Proton `NotifyResourceStateChange` API を使用してこれらの非同期通知を送信する必要があります AWS Proton。

セルフマネージドプロビジョニングは、コンソールまたは AWS CLIを通して使用できます。以下の例では、Terraform でセルフマネージドプロビジョニングを使用する方法を示します。

AWS マネジメントコンソール

コンソールでセルフマネージドプロビジョニングを使用して Terraform 環境を作成します。

1. AWS Proton コンソールで、[環境] を選択します。

2. [Create environment (環境の作成)] を選択します。
3. [Choose an environment template (環境テンプレートを選択する)] ページで、Terraform テンプレートを選択して [Configure (設定)] を選択します。
4. [Configure environment (環境を設定する)] ページの [Provisioning] (プロビジョニング)] セクションで [Self-managed provisioning (セルフマネージドプロビジョニング)] を選択します。
5. [プロビジョニングリポジトリの詳細] セクション:

   a. プロビジョニングリポジトリをまだリンク AWS Protonしていない場合は、新しいリポジトリを選択し、リポジトリプロバイダーの 1 つを選択し、CodeStar 接続の場合は接続の 1 つを選択します。

   > **Note**
   >
   > 関連するリポジトリプロバイダーアカウントにまだ接続していない場合は、[新しい CodeStar 接続を追加] を選択します。次に、接続を作成し、[CodeStar 接続] メニューの横にある更新ボタンを選択します。これで、あなたの新しい接続をメニューから選択できます。

   リポジトリを既にリンクしている場合は AWS Proton、「既存のリポジトリ」を選択します。

   b. [Repository name (リポジトリ名)] で、リポジトリを選択します。ドロップダウンメニューには、既存のリポジトリの場合はリンクされたリポジトリ、新規リポジトリの場合はプロバイダーアカウントのリポジトリのリストが表示されます。

   c. [ブランチ名] で、リポジトリブランチの 1 つを選択します。
6. [Environment Settings (環境の設定)] セクションで [Environment name] に環境名を入力します。
7. (オプション) この環境についての説明を入力します。
8. (オプション) [Tags (タグ)] セクションで [Add new tag (新しいタグを追加)] を選択し、キーと値を入力してカスタマーマネージドタグを作成します。
9. [Next (次へ)] を選択します。
10. [Configure environment custom settings (環境カスタム設定の構成)] ページでは、`required` パラメータの値を入力してください。`optional` パラメータの値を入力するか、表示されるデフォルト値を使用します。
11. [Next (次へ)] を選択して入力内容を見直します。

12. [Create (作成)] を選択してプルリクエストを送信します。

    - プルリクエストを承認すると、デプロイが進行中になります。
    - プルリクエストを拒否すると、環境の作成はキャンセルされます。
    - プルリクエストがタイムアウトになった場合、環境の作成は完了しません。

13. 環境の詳細とステータス、および環境の AWS マネージドタグとカスタマーマネージドタグを表示します。

14. ナビゲーションペインで [Environments (環境)] を選択します。

    新しいページには、ステータスやその他の環境の詳細とともに環境のリストが表示されます。

AWS CLI

セルフマネージドプロビジョニングを実施する環境を作成するときは、`provisioningRepository` パラメータを追加し、パラメータ`ProtonServiceRoleArn` と `environmentAccountConnectionId` パラメータを省略します。

を使用して AWS CLI 、セルフマネージドプロビジョニングで Terraform 環境を作成します。

1. 環境を作成し、見直しと承認の対象としてリポジトリにプルリクエストを送信します。

   次の例では、環境テンプレートスキーマファイルに基づいて 2 つの入力値を定義する YAML 形式の仕様ファイルを示します。`get-environment-template-minor-version` コマンドを実行して、環境テンプレートスキーマを表示します。

   仕様:

   ```
   proton: EnvironmentSpec
   spec:
     ssm_parameter_value: "test"
   ```

   次のコマンドを実行して、環境を作成します。

   ```
   $ aws proton create-environment \
       --name "pr-environment" \
       --template-name "pr-env-template" \
       --template-major-version "1" \
   ```

```
    --provisioning-repository="branch=main,name=myrepos/env-
repo,provider=GITHUB" \
    --spec "file://env-spec.yaml"
```

レスポンス:>

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/pr-
environment",
        "createdAt": "2021-11-18T17:06:58.679000+00:00",
        "deploymentStatus": "IN_PROGRESS",
        "lastDeploymentAttemptedAt": "2021-11-18T17:06:58.679000+00:00",
        "name": "pr-environment",
        "provisioningRepository": {
            "arn": "arn:aws:proton:region-id:123456789012:repository/
github:myrepos/env-repo",
            "branch": "main",
            "name": "myrepos/env-repo",
            "provider": "GITHUB"
        },
        "templateName": "pr-env-template"
    }
```

2. リクエストを見直します。

   - リクエストを承認すると、プロビジョニングが進行中になります。
   - リクエストを拒否すると、環境の作成はキャンセルされます。
   - プルリクエストがタイムアウトになった場合、環境の作成は完了しません。

3. プロビジョニングステータスを非同期的に に提供します AWS Proton。次の の例では AWS Proton 、プロビジョニングが成功したことを に通知します。

```
$ aws proton notify-resource-deployment-status-change \
    --resource-arn "arn:aws:proton:region-id:123456789012:environment/pr-
environment" \
    --status "SUCCEEDED"
```

環境データを表示する

AWS Proton コンソールまたは を使用して、環境の詳細データを表示できます AWS CLI。

AWS マネジメントコンソール

AWS Proton コンソールを使用して、詳細を含む環境のリストと、詳細データを含む個々の環境のリストを表示できます。

1. 環境のリストを表示するには、ナビゲーションペインで [Environments (環境)] を選択します。
2. 詳細データを表示するには、環境の名前を選択します。

   環境の詳細データを表示します。

AWS CLI

環境の詳細 AWS CLI を取得または一覧表示します。

次のコマンドを実行します。

```
$ aws proton get-environment \
    --name "MySimpleEnv"
```

レスポンス:

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv",
        "createdAt": "2020-11-11T23:03:05.405000+00:00",
        "deploymentStatus": "SUCCEEDED",
        "lastDeploymentAttemptedAt": "2020-11-11T23:03:05.405000+00:00",
        "lastDeploymentSucceededAt": "2020-11-11T23:03:05.405000+00:00",
        "name": "MySimpleEnv",
        "protonServiceRoleArn": "arn:aws:iam::123456789012:role/ProtonServiceRole",
        "spec": "proton: EnvironmentSpec\nspec:\n  my_sample_input: \"the first\"\n
 my_other_sample_input: \"the second\"\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "0",
        "templateName": "simple-env"
    }
}
```

環境を更新する

AWS Proton 環境が環境アカウント接続に関連付けられている場合は、protonServiceRoleArnパラメータを更新または含めて、環境アカウント接続を更新または接続しないでください。

新しい環境アカウント接続に更新できるのは、次の両方に当てはまる場合にのみです。

- 環境アカウント接続は、現在の環境アカウント接続を作成したときと同じ環境アカウントで作成しました。
- >環境アカウント接続は、現在の環境に関連付けられています。

環境が環境アカウント接続に関連付けられていない場合、environmentAccountConnectionId パラメータを更新したり含めたりしないでください。

environmentAccountConnectionId または protonServiceRoleArn のパラメータと値を更新できます。両方を更新することはできません。

環境でセルフマネージドプロビジョニングを使用する場合、provisioning-repository パラメータを更新せずに environmentAccountConnectionId パラメータおよび protonServiceRoleArn パラメータを省略してください。

環境を更新しようとする場合、以下に挙げるように 4 つのモードがあります。を使用する場合 AWS CLI、 deployment-typeフィールドは モードを定義します。コンソールを使用する場合、これらのモードは [Actions (アクション)] ドロップダウンリストに表示される [Edit (編集)] 、[Update (更新)]、[Update minor (マイナー更新)] 、および [Update major (メジャー更新)] にマップされます。

NONE

このモードでは、デプロイは発生しません。リクエストしたメタデータパラメータのみが更新されます。

CURRENT_VERSION

このモードでは、指定した新しい仕様で環境がデプロイされ更新されます。リクエストしたパラメータのみが更新されます。この deployment-type を使用する場合、マイナーバージョンまたはメジャーバージョンのパラメータを含めないでください。

`MINOR_VERSION`

このモードでは、環境がデプロイされ、デフォルトで使用中の現在のメジャーバージョンの公開され推奨される (最新の) マイナーバージョンで更新されます。また、使用中の現在のメジャーバージョンについて別のマイナーバージョンを指定することもできます。

`MAJOR_VERSION`

このモードでは、環境がデプロイされ、デフォルトで現在のテンプレートの公開され推奨される (最新の) メジャーとマイナーバージョンで更新されます。使用中のメジャーバージョンやマイナーバージョン (オプション) よりも高いメジャーバージョンも指定できます。

トピック

AWS マネージドプロビジョニング環境を更新する

標準プロビジョニングは CloudFormationでプロビジョニングする環境によってのみサポートされます。

コンソールまたは AWS CLI を使用して環境を更新します。

AWS マネジメントコンソール

以下に示す手順に従って、コンソールで環境を更新します。

1. 次の 2 つのステップのうちの 1 つを選択します。

   a. 環境のリストで、以下の操作をします。

      i. AWS Proton コンソールで、[環境] を選択します。
      ii. 環境のリストで、更新したい環境の左にあるラジオボタンを選択します。

   b. コンソール環境の詳細ページで以下の操作をします。

      i. AWS Proton コンソールで、[環境] を選択します。
      ii. 環境のリストで、更新したい環境の名前を選択します。

2. 次の 4 つのステップのいずれかを選択して環境を更新します。

   a. 環境デプロイを要求しない編集を加えるには

      i. たとえば、説明を変更するには、次のように操作します。

         [Edit (編集)] を選択します。

      ii. フォームに必要な値を入力して [Next (次へ)] を選択します。

      iii. 編集内容を見直して [Update (更新)] を選択します。

   b. メタデータ入力のみを更新するには

      i. [Actions (アクション)] を選択してから [Update (更新)] を選択します。

      ii. フォームに必要な値を入力して [Edit (編集)] を選択します。

      iii. フォームに必要な値を入力し、[Review (確認)]ページが表示されるまで [Next (次へ)] を選択します。

      iv. 更新内容を見直して [Update (更新)] を選択します。

   c. 環境テンプレートの新しいマイナーバージョンを更新するには

      i. [Actions (アクション)] を選択してから [Update minor (マイナー更新)] を選択します。

      ii. フォームに必要な値を入力して [Next (次へ)] を選択します。

      iii. フォームに必要な値を入力し、[Review (確認)] ページが表示されるまで [Next (次へ)] を選択します。

      iv. 更新内容を見直して [Update (更新)] を選択します。

   d. 環境テンプレートの新しいメジャーバージョンを更新するには

      i. [Actions (アクション)] を選択してから [Update major (メジャー更新)] を選択します。

      ii. フォームに必要な値を入力して [Next (次へ)] を選択します。

      iii. フォームに必要な値を入力し、[Review (確認)] ページが表示されるまで [Next (次へ)] を選択します。

      iv. 更新内容を見直して [Update (更新)] を選択します。

AWS CLI

AWS を使用して、環境を新しいマイナーバージョンに AWS Proton AWS CLI 更新します。

次のコマンドを入力して環境を更新します。

```
$ aws proton update-environment \
        --name "MySimpleEnv" \
        --deployment-type "MINOR_VERSION" \
        --template-major-version "1" \
        --template-minor-version "1" \
        --proton-service-role-arn arn:aws:iam::123456789012:role/service-
role/ProtonServiceRole \
        --spec "file:///spec.yaml"
```

レスポンス:

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv",
        "createdAt": "2021-04-02T17:29:55.472000+00:00",
        "deploymentStatus": "IN_PROGRESS",
        "lastDeploymentAttemptedAt": "2021-04-02T17:48:26.307000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T17:29:55.472000+00:00",
        "name": "MySimpleEnv",
        "protonServiceRoleArn": "arn:aws:iam::123456789012:role/service-role/
ProtonServiceRole",
        "templateMajorVersion": "1",
        "templateMinorVersion": "0",
        "templateName": "simple-env"
    }
}
```

次のコマンドを実行して、ステータスを取得して確認します。

```
$ aws proton get-environment \
        --name "MySimpleEnv"
```

レスポンス:

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv",
        "createdAt": "2021-04-02T17:29:55.472000+00:00",
        "deploymentStatus": "SUCCEEDED",
```

```
        "environmentName": "MySimpleEnv",
        "lastDeploymentAttemptedAt": "2021-04-02T17:48:26.307000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T17:48:26.307000+00:00",
        "protonServiceRoleArn": "arn:aws:iam::123456789012:role/service-role/
ProtonServiceRole",
        "spec": "proton: EnvironmentSpec\n\nspec:\n  my_sample_input: hello\n
 my_other_sample_input: everybody\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "simple-env"
    }
}
```

セルフマネージドプロビジョニング環境の更新

セルフマネージドプロビジョニングをサポートするのは、Terraform でプロビジョニングする環境のみです。

コンソールまたは AWS CLI を使用して環境を更新します。

AWS マネジメントコンソール

以下に示す手順に従って、コンソールで環境を更新します。

1. 次の 2 つのステップのうちの 1 つを選択します。

   a. 環境のリストで、以下の操作をします。

      i. AWS Proton コンソールで、[環境] を選択します。

      ii. 環境のリストで、更新したい環境テンプレートの左にあるラジオボタンを選択します。

   b. コンソール環境の詳細ページで以下の操作をします。

      i. AWS Proton コンソールで、[環境] を選択します。

      ii. 環境のリストで、更新したい環境の名前を選択します。

2. 次の 4 つのステップのいずれかを選択して環境を更新します。

   a. 環境デプロイを要求しない編集を加えるには

      i. たとえば、説明を変更するには、次のように操作します。

[Edit (編集)] を選択します。

ii. フォームに必要な値を入力して [Next (次へ)] を選択します。

iii. 編集内容を見直して [Update (更新)] を選択します。

b. メタデータ入力のみを更新するには

i. [Actions (アクション)] を選択してから [Update (更新)] を選択します。

ii. フォームに必要な値を入力して [Edit (編集)] を選択します。

iii. フォームに必要な値を入力し、[Review (確認)]ページが表示されるまで [Next (次へ)] を選択します。

iv. 更新内容を見直して [Update (更新)] を選択します。

c. 環境テンプレートの新しいマイナーバージョンを更新するには

i. [Actions (アクション)] を選択してから [Update minor (マイナー更新)] を選択します。

ii. フォームに必要な値を入力して [Next (次へ)] を選択します。

iii. フォームに必要な値を入力し、[Review (確認)] ページが表示されるまで [Next (次へ)] を選択します。

iv. 更新内容を見直して [Update (更新)] を選択します。

d. 環境テンプレートの新しいメジャーバージョンを更新するには

i. [Actions (アクション)] を選択してから [Update major (メジャー更新)] を選択します。

ii. フォームに必要な値を入力して [Next (次へ)] を選択します。

iii. フォームに必要な値を入力し、[Review (確認)] ページが表示されるまで [Next (次へ)] を選択します。

iv. 更新内容を見直して [Update (更新)] を選択します。

AWS CLI

AWS CLI を使用して、セルフマネージドプロビジョニングで Terraform 環境を新しいマイナーバージョンに更新します。

1. 次のコマンドを入力して環境を更新します。

```
$ aws proton update-environment \
    --name "pr-environment" \
    --deployment-type "MINOR_VERSION" \
    --template-major-version "1" \
    --template-minor-version "1" \
    --provisioning-repository "branch=main,name=myrepos/env-
repo,provider=GITHUB" \
    --spec "file://env-spec-mod.yaml"
```

レスポンス:

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/pr-
environment",
        "createdAt": "2021-11-18T21:09:15.745000+00:00",
        "deploymentStatus": "IN_PROGRESS",
        "lastDeploymentAttemptedAt": "2021-11-18T21:25:41.998000+00:00",
        "lastDeploymentSucceededAt": "2021-11-18T21:09:15.745000+00:00",
        "name": "pr-environment",
        "provisioningRepository": {
            "arn": "arn:aws:proton:region-id:123456789012:repository/
github:myrepos/env-repo",
            "branch": "main",
            "name": "myrepos/env-repo",
            "provider": "GITHUB"
        },
        "templateMajorVersion": "1",
        "templateMinorVersion": "0",
        "templateName": "pr-env-template"
    }
}
```

2. 次のコマンドを実行して、ステータスを取得して確認します。

```
$ aws proton get-environment \
    --name "pr-environment"
```

レスポンス:

```
{
```

```
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/pr-
environment",
        "createdAt": "2021-11-18T21:09:15.745000+00:00",
        "deploymentStatus": "SUCCEEDED",
        "lastDeploymentAttemptedAt": "2021-11-18T21:25:41.998000+00:00",
        "lastDeploymentSucceededAt": "2021-11-18T21:25:41.998000+00:00",
        "name": "pr-environment",
        "provisioningRepository": {
            "arn": "arn:aws:proton:region-id:123456789012:repository/
github:myrepos/env-repo",
            "branch": "main",
            "name": "myrepos/env-repo",
            "provider": "GITHUB"
        },
        "spec": "proton: EnvironmentSpec\nspec:\n   ssm_parameter_value: \"test
\"\n ssm_another_parameter_value: \"update\"\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "pr-env-template"
    }
}
```

3. によって送信されたプルリクエストを確認します AWS Proton。

   - リクエストを承認すると、プロビジョニングが進行中になります。
   - リクエストを拒否すると、環境の作成はキャンセルされます。
   - プルリクエストがタイムアウトになった場合、環境の作成は完了しません。

4. プロビジョニングステータスを に提供します AWS Proton。

```
$ aws proton notify-resource-deployment-status-change \
    --resource-arn "arn:aws:proton:region-id:123456789012:environment/pr-
environment" \
    --status "SUCCEEDED"
```

進行中の環境デプロイをキャンセルする

deploymentStatus が IN_PROGRESS. AWS Proton attempts にある場合、環境更新デプロイをキャンセルしようとできます。正常にキャンセルされる保証はありません。

更新デプロイをキャンセルすると、 は次のステップに示すようにデプロイをキャンセル AWS Proton しようとします。

AWSマネージドプロビジョニングでは、以下 AWS Proton を実行します。

- デプロイの状態を CANCELLING に設定します。
- 進行中のデプロイを停止し、IN_PROGRESS の際にデプロイによって作成された新しいリソースを削除します。
- デプロイの状態を CANCELLED に設定します。
- リソースの状態をデプロイが開始される前の状態に戻します。

セルフマネージドプロビジョニングでは、 は次の AWS Proton 操作を行います。

- リポジトリに変更をマージしないように、プルリクエストを閉じようとします。
- プルリクエストが正常に閉じた場合、デプロイの状態を CANCELLED に設定します。

環境デプロイをキャンセルする手順については、『AWS Proton API リファレンス』の「CancelEnvironmentDeployment」を参照してください。

コンソールまたは CLI を使用して、進行中の環境をキャンセルできます。

AWS マネジメントコンソール

以下に示す手順に従って、コンソールで環境更新デプロイをキャンセルします。

1. AWS Proton コンソールのサービスナビゲーションペインで [Environments (環境)] を選択します。
2. 環境のリストで、デプロイの更新をキャンセルしたい環境の名前を選択します。
3. 更新のデプロイステータスが [In progress (進行中)] の場合、環境の詳細ページで [Action (アクション)] を選択してから [Cancel deployment (デプロイをキャンセル)] を選択します。
4. キャンセルするかどうかの確認を求めるモーダルが表示されます。[Cancel deployment (デプロイをキャンセル)] を選択します。
5. 更新のデプロイステータスが [Cancelling (キャンセル中)] に変わり、キャンセルが完了すると [Cancelled (キャンセル済み)] に変わります。

AWS CLI

を使用して、新しいマイナーバージョン 2 への IN_PROGRESS 環境更新デプロイ AWS Proton AWS CLI をキャンセルします。

この例で使用するテンプレートには待機条件が含まれており、更新のデプロイが成功する前にキャンセルが開始されます。

次のコマンドを実行して更新をキャンセルします。

```
$ aws proton cancel-environment-deployment \
        --environment-name "MySimpleEnv"
```

レスポンス:

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv",
        "createdAt": "2021-04-02T17:29:55.472000+00:00",
        "deploymentStatus": "CANCELLING",
        "lastDeploymentAttemptedAt": "2021-04-02T18:15:10.243000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T17:48:26.307000+00:00",
        "name": "MySimpleEnv",
        "protonServiceRoleArn": "arn:aws:iam::123456789012:role/service-role/
ProtonServiceRole",
        "spec": "proton: EnvironmentSpec\n\nspec:\n  my_sample_input: hello\n
 my_other_sample_input: everybody\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "simple-env"
    }
}
```

次のコマンドを実行して、ステータスを取得して確認します。

```
$ aws proton get-environment \
        --name "MySimpleEnv"
```

レスポンス:

```
{
```

```
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv",
        "createdAt": "2021-04-02T17:29:55.472000+00:00",
        "deploymentStatus": "CANCELLED",
        "deploymentStatusMessage": "User initiated cancellation.",
        "lastDeploymentAttemptedAt": "2021-04-02T18:15:10.243000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T17:48:26.307000+00:00",
        "name": "MySimpleEnv",
        "protonServiceRoleArn": "arn:aws:iam::123456789012:role/service-role/
ProtonServiceRole",
        "spec": "proton: EnvironmentSpec\n\nspec:\n  my_sample_input: hello\n
 my_other_sample_input: everybody\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "simple-env"
    }
}
```


環境を削除する

AWS Proton コンソールまたは を使用して AWS Proton 環境を削除できます AWS CLI。

> **Note**
>
> コンポーネントが関連付けられている環境は削除できません。このような環境を削除するには、まずその環境で実行中のコンポーネントをすべて削除する必要があります。コンポーネントの詳細については、「コンポーネント」を参照してください。

AWS マネジメントコンソール

次の 2 つのオプションで説明するように、コンソールを使用して環境を削除します。

環境のリストで、以下の操作をします。

1. AWS Proton コンソールで、[環境] を選択します。
2. 環境のリストで、削除したい環境の左にあるラジオボタンを選択します。
3. [Actions (アクション)] を選択してから [Delete (削除)] を選択します。
4. モーダルから Delete アクションの確認を求めるプロンプトが表示されます。

5. 指示に従って操作し、[Yes, delete (はい、削除します)] を選択します。

環境の詳細ページで以下の操作をします。

1. AWS Proton コンソールで、[環境] を選択します。
2. 環境のリストで、削除したい環境の名前を選択します。
3. 関豹の詳細ページで [Actions (アクション)] を選択してから [Delete (削除)] を選択します。
4. モーダルから消去アクションの確認を求めるプロンプトが表示されます。
5. 指示に従って操作し、[Yes, delete (はい、削除します)] を選択します。

AWS CLI

を使用して環境 AWS CLI を削除します。

サービスまたはサービスインスタンスが環境にデプロイされている場合、環境を削除しないでください。

次のコマンドを実行します。

```
$ aws proton delete-environment \
    --name "MySimpleEnv"
```

レスポンス:

```
{
    "environment": {
        "arn": "arn:aws:proton:region-id:123456789012:environment/MySimpleEnv",
        "createdAt": "2021-04-02T17:29:55.472000+00:00",
        "deploymentStatus": "DELETE_IN_PROGRESS",
        "lastDeploymentAttemptedAt": "2021-04-02T17:48:26.307000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T17:48:26.307000+00:00",
        "name": "MySimpleEnv",
        "protonServiceRoleArn": "arn:aws:iam::123456789012:role/ProtonServiceRole",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "simple-env"
    }
}
```

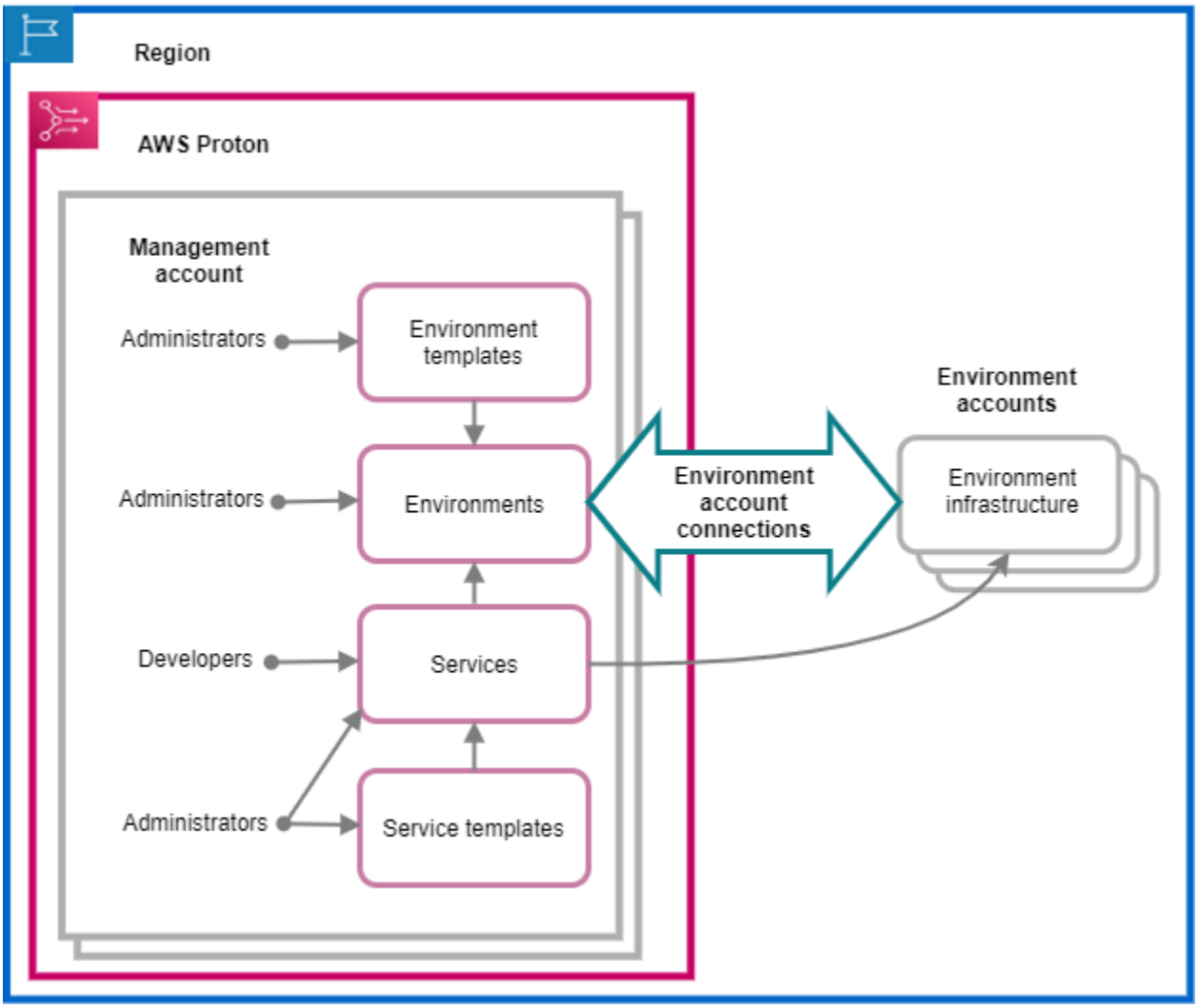
環境アカウント接続

概要:

あるアカウントで AWS Proton 環境を作成および管理し、別のアカウントでそのインフラストラクチャリソースをプロビジョニングする方法について説明します。これにより、可視性と効率性が大幅に向上します。環境アカウント接続では、 CloudFormation Infrastructure as Code を含む標準プロビジョニングのみがサポートされます。

> **Note**
>
> このトピックの情報は、AWS マネージドプロビジョニングが設定されている環境に関する内容です。セルフマネージドプロビジョニングで設定された環境では、 AWS Proton はインフラストラクチャを直接プロビジョニングしません。代わりに、プルリクエスト (PR) をあなたのリポジトリに送信してプロビジョニングします。自動コードに正しい ID とロールを引き受けさせる責任はあなたにあります。
> プロビジョニング方法の詳細については、「the section called “プロビジョニングの方法”」を参照してください。

用語



AWS Proton 環境アカウント接続を使用すると、あるアカウントから環境を作成し AWS Proton 、そのインフラストラクチャを別のアカウントにプロビジョニングできます。

管理アカウント

管理者として、別の AWS Proton 環境アカウントにインフラストラクチャリソースをプロビジョニングする環境を作成する単一のアカウント。

環境アカウント

別のアカウント内で AWS Proton 環境を作成した際に環境インフラストラクチャがプロビジョニングされるアカウント。

環境アカウント接続

管理アカウントと環境アカウントの間をつなぐセキュアな双方向接続。次のセクションで詳しく説明するように、認可と権限が維持されます。

環境アカウントで環境アカウント接続を作成すると、特定のリージョンで環境アカウント接続の表示と使用ができるのは、同じリージョン内の管理アカウントのみです。つまり、管理アカウントで作成された AWS Proton 環境と環境アカウントでプロビジョニングされた環境インフラストラクチャは、同じリージョンに存在する必要があります。

環境アカウントの接続に関する考慮事項

- 環境アカウントでプロビジョニングする環境ごとに環境アカウント接続が必要です。
- 環境アカウント接続クォータの詳細については、「AWS Proton クォータ」を参照してください。

タグ付け

環境アカウントで、 コンソールまたは AWS CLI を使用して、環境アカウント接続のカスタマーマネージドタグを表示および管理します。 AWS マネージドタグは、環境アカウント接続では生成されません。詳細については、「Tagging」を参照してください。

1 つのアカウントに環境を作成し、別のアカウントでそのインフラストラクチャをプロビジョニングします。

単一の管理アカウントから環境を作成してプロビジョニングするには、作成しようとする環境のための環境アカウントを設定します。

環境アカウント内で起動して接続を作成します。

環境アカウントで、環境インフラストラクチャリソースのプロビジョニングに必要なアクセス許可のみを対象とする AWS Proton サービスロールを作成します。詳細については、「AWS Proton を使用してプロビジョニングするための サービスロール CloudFormation」を参照してください。

次いで、環境アカウント接続リクエストを作成してあなたの管理アカウントに送信します。リクエストが承諾されると、 AWS Proton は、関連付けられた環境アカウントで環境リソースのプロビジョニングを許可する関連付けられた IAM ロールを使用できます。

管理アカウント内で、環境アカウント接続を受け入れるか拒否します。

管理アカウント内で、環境アカウント接続リクエストを受け入れるか拒否します。管理アカウントから環境アカウント接続を削除することはできません。

リクエストを受け入れると、 AWS Proton は関連付けられた環境アカウントでリソースのプロビジョニングを許可する関連付けられた IAM ロールを使用できます。

環境インフラストラクチャリソースは、関連付けられた環境アカウントにプロビジョニングされます。 AWS Proton APIs は、管理アカウントから環境とそのインフラストラクチャリソースへのアクセスと管理にのみ使用できます。詳細については、「1 つのアカウントに環境を作成し、別のアカウントでプロビジョニングします。」および「環境を更新する」を参照してください。

リクエストを拒否した後は、拒否した環境アカウント接続を受け入れたり使用することはできません。

> **Note**
>
> 環境に接続されている環境アカウント接続を拒否することはできません。環境アカウント接続を拒否するには、まず関連付けられた環境を削除する必要があります。

環境アカウント内で、プロビジョニングされたインフラストラクチャリソースにアクセスします。

環境アカウント内で、プロビジョニングされたインフラストラクチャリソースの表示やアクセスが可能です。たとえば、CloudFormation API アクションを使用して、必要に応じてスタックをモニタリングおよびクリーンアップできます。 AWS Proton API アクションを使用して、インフラストラクチャリソースのプロビジョニングに使用された AWS Proton 環境にアクセスまたは管理することはできません。

環境アカウント内で作成した環境アカウント接続を環境アカウント内で削除できます。受け入れも拒否もできません。環境によって使用されている AWS Proton 環境アカウント接続を削除すると、AWS Proton は、環境アカウントと名前付き環境に対して新しい環境接続が受け入れられるまで、環境インフラストラクチャリソースを管理できなくなります。プロビジョニングされて環境接続がないまま残っているリソースをクリーンアップする責任はあなたにあります。

コンソールまたは CLI を使用して環境アカウント接続を管理する

コンソールまたは CLI を使用して、環境アカウント接続を作成および管理できます。

AWS マネジメントコンソール

コンソールを使用して環境アカウント接続を作成し、次の手順に示すように管理アカウントにリクエストを送信します。

1. 管理アカウント内で作成する予定の環境の名前を決めるか、または環境アカウント接続を必要とする既存の環境の名前を選択します。

2. 環境アカウント内の AWS Proton コンソールでナビゲーションペインにある [Environment account connections (環境アカウント接続)] を選択します。
3. [Environment account connections (環境アカウント接続)] ページで[Request to connect (接続のリクエスト)] を選択します。

> **Note**
>
> [Environment account connection (環境アカウント接続ページ)] の見出しに挙がっているアカウント ID を確認してください。名前付き環境をプロビジョニングする環境アカウントのアカウント ID と一致していることを確認してください。

4. [Request to connect (接続のリクエスト)] ページで以下の操作をします。
   a. [Connect to management account] (管理アカウントに接続する) セクションで [Management account ID] と [Environment name] にステップ 1 で入力した管理アカウント ID と環境名を入力します。
   b. 環境ロールセクションで、新しいサービスロールを選択すると、新しいロール AWS Proton が自動的に作成されます。または、[Existing service role (既存のサービスロール)] と以前に作成したサービスロールの名前を選択します。

      > **Note**
      >
      > AWS Proton が自動的に作成するロールには、幅広いアクセス許可があります。ロールをスコープダウンして、環境インフラストラクチャリソースのプロビジョニングに必要な権限のみにすることをお勧めします。詳細については、「AWS Proton を使用してプロビジョニングするための サービスロール CloudFormation」を参照してください。

   c. (オプション) [Tags (タグ)] セクションで、[Add new tags (新しいタグを追加)] を選択して、環境アカウント接続用のカスタマーマネージドタグを作成します。
   d. [Request to connect (接続のリクエスト)]を選択します。
5. リクエストは [Environment connections sent to a management account (管理アカウントに送信される環境接続)] テーブルに [Pending] (保留中) として表示され、モーダルを見ると管理アカウントからのリクエストを受け入れる方法がわかります。

環境アカウント接続リクエストを受け入れるか拒否します。

1. 管理アカウント内の AWS Proton コンソールでナビゲーションペインにある [Environment account connections (環境アカウント接続)] を選択します。
2. [Environment account connections (環境アカウント接続)] ページの [Environment account connection requests (環境アカウント接続リクエスト)] テーブルで、承諾または拒否する環境接続リクエストを選択します。

> **Note**
>
> [Environment account connection (環境アカウント接続ページ)] の見出しに挙がっているアカウント ID を確認してください。拒否する環境アカウント接続に関連付けられている管理アカウントのアカウント ID と一致していることを確認してください。この環境アカウント接続を拒否した後は、拒否した環境アカウント接続の受け入れや使用はできません。

3. [Reject (拒否)] または [Accept (承諾)] を選択します。
   - [Reject (拒否)] を選択した場合、ステータスが [Pending (保留中)] から [Rejected (拒否)] に変わります。
   - [Accept] (承諾) を選択した場合、ステータスが [Pending (保留中)] から [Connected (接続済み)] に変わります。

環境アカウント接続を削除します。

1. 環境アカウント内の AWS Proton コンソールでナビゲーションペインにある [Environment account connections (環境アカウント接続)]を選択します。

> **Note**
>
> [Environment account connection (環境アカウント接続ページ)] の見出しに挙がっているアカウント ID を確認してください。拒否する環境アカウント接続に関連付けられている管理アカウントのアカウント ID と一致していることを確認してください。この環境アカウント接続を削除すると、 AWS Proton は環境アカウントの環境インフラストラクチャリソースを管理できません。管理できるのは、環境アカウントと名前付き環境のための新しい環境アカウント接続が管理アカウントによって受け入れられた後のみです。

2. [Environment account connections(環境アカウント接続)] ページで [Sent requests to connect to management account (管理アカウントに送信した接続リクエスト)] セクションで、[Delete (削除)] を選択します。
3. モーダルから消去アクションの確認を求めるプロンプトが表示されます。[Delete (削除)] をクリックします。

AWS CLI

管理アカウント内で作成する予定の環境の名前を決めるか、または環境アカウント接続を必要とする既存の環境の名前を選択します。

環境アカウント内で環境アカウントを作成します。

次のコマンドを実行します。

```
$ aws proton create-environment-account-connection \
    --environment-name "simple-env-connected" \
    --role-arn "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role" \
    --management-account-id "111111111111"
```

レスポンス:

```
{
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:region-id:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-connected",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:13:50.847000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role",
        "status": "PENDING"
    }
}
```

次のコマンドとレスポンスに示すように、管理アカウント内で環境アカウント接続を承認または拒否します。

> **Note**
>
> この環境アカウント接続を拒否すると、拒否した環境アカウント接続の受け入れや使用はできなくなります。

[Reject (拒否)] を指定した場合、ステータスが [Pending (保留中)] から [Rejected (拒否)] に変わります。

[Accept (承諾)] を指定した場合、ステータスが [Pending (保留中)] から [Connected (接続済み)] に変わります。

次のコマンドを実行して、環境アカウント接続を受け入れます。

```
$ aws proton accept-environment-account-connection \
    --id "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

レスポンス:

```
{
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:region-id:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-connected",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:15:33.486000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role",
        "status": "CONNECTED"
    }
}
```

次のコマンドを実行して、環境アカウント接続を拒否します。

```
$ aws proton reject-environment-account-connection \
    --id "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

レスポンス:

```
{
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:us-east-1:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "status": "REJECTED",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-reject",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:13:50.847000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role"
    }
}
```

環境アカウント接続を表示します。環境アカウント接続を取得または一覧表示できます。

次の get コマンドを実行します。

```
$ aws proton get-environment-account-connection \
    --id "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

レスポンス:

```
{
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:region-id:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-connected",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:15:33.486000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role",
        "status": "CONNECTED"
    }
}
```

環境アカウント内で環境アカウントを削除します。

> **Note**
>
> この環境アカウント接続を削除する AWS Proton と、環境アカウントと名前付き環境に対して新しい環境接続が受け入れられるまで、環境アカウントの環境インフラストラクチャリソースを管理できなくなります。プロビジョニングされて環境接続がないまま残っているリソースをクリーンアップする責任はあなたにあります。

次のコマンドを実行します。

```
$ aws proton delete-environment-account-connection \
    --id "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
```

レスポンス:

```
{
    "environmentAccountConnection": {
        "arn": "arn:aws:proton:us-east-1:222222222222:environment-account-
connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "environmentAccountId": "222222222222",
        "environmentName": "simple-env-connected",
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "lastModifiedAt": "2021-04-28T23:13:50.847000+00:00",
        "managementAccountId": "111111111111",
        "requestedAt": "2021-04-28T23:13:50.847000+00:00",
        "roleArn": "arn:aws:iam::222222222222:role/service-role/env-account-proton-
service-role",
        "status": "CONNECTED"
    }
}
```

カスタマーマネージド環境

カスタマーマネージド環境では、 AWS Proton 環境としてデプロイ済みの VPC などの既存のインフラストラクチャを使用できます。カスタマーマネージド環境を使用している場合は、 の外部で独自の共有リソースをプロビジョニングできます AWS Proton。ただし、デプロイ時に が関連するプロビジョニング出力 AWS Proton を AWS Proton サービスの入力として消費することはできます。出力が変更できる場合、 AWS Proton は更新を受け入れることができます。プロビジョニング AWS Proton は の外部で管理されるため、 は環境を直接変更できません AWS Proton。

環境を作成したら、Amazon ECS クラスター名や Amazon VPC IDs などの環境 AWS Proton を作成 AWS Proton した場合と同じ出力を に提供する責任があります。

この機能を使用すると、 AWS Proton サービステンプレートから AWS Proton この環境にサービスリソースをデプロイおよび更新できます。ただし、環境自体はテンプレートの更新によって変更されません AWS Proton。環境の更新を実行し、それらの出力を更新するのはお客様の責任です AWS Proton。

管理環境とカスタマー AWS Proton 管理環境が混在する複数の環境を 1 つの アカウントに設定できます。2 番目のアカウントをリンクし、プライマリアカウントの AWS Proton テンプレートを使用して、2 番目のリンクされたアカウントで環境とサービスへのデプロイと更新を実行することもできます。

カスタマーマネージドキーの使用方法

管理者が最初に行う必要があるのは、インポートされたカスタマーマネージド環境テンプレートを登録することです。テンプレートバンドルにはマニフェストやインフラストラクチャファイルを提供しないでください。スキーマのみを指定してください。

以下のスキーマは、オープン API 形式を使用した出力のリストの概要を示し、 CloudFormation テンプレートからの出力をレプリケートします。

> **Important**
>
> 出力には文字列入力のみが許可されます。

次の例は、対応する Fargate CloudFormation テンプレートの テンプレートの出力セクションのスニペットです。

```
Outputs:
  ClusterName:
    Description: The name of the ECS cluster
    Value: !Ref 'ECSCluster'
  ECSTaskExecutionRole:
    Description: The ARN of the ECS role
    Value: !GetAtt 'ECSTaskExecutionRole.Arn'
  VpcId:
    Description: The ID of the VPC that this stack is deployed in
    Value: !Ref 'VPC'
[...]
```

対応する AWS Proton インポートされた環境のスキーマは次のようになります。スキーマにはデフォルト値を指定しないでください。

```
schema:
  format:
    openapi: "3.0.0"
  environment_input_type: "EnvironmentOutput"
  types:
    EnvironmentOutput:
      type: object
      description: "Outputs of the environment"
      properties:
        ClusterName:
          type: string
          description: "The name of the ECS cluster"
        ECSTaskExecutionRole:
          type: string
          description: "The ARN of the ECS role"
        VpcId:
          type: string
          description: "The ID of the VPC that this stack is deployed in"
[...]
```

テンプレートの登録時に、このテンプレートがインポートされ、バンドルの Amazon S3 バケットの場所を提供します。 は、テンプレートをドラフトに入れる前に、スキーマに CloudFormation テンプレートパラメータのみが含まれ`environment_input_type`、テンプレートパラメータが含まれていないこと AWS Proton を検証します。

インポートした環境を作成するには、以下の情報を指定します。

- デプロイ時に使用する IAM ロール。
- 必要な出力の値を含む仕様。

これらの両方は、コンソールまたは を通じて、通常の環境のデプロイと同様のプロセス AWS CLI を使用して提供できます。

CodeBuild プロビジョニングロールの作成

CloudFormation や Terraform などの Infrastructure as a Code (IaaC) ツールには、さまざまなタイプの AWS リソースに対するアクセス許可が必要です。例えば、IaaC テンプレートで Amazon S3 バ

ケットを宣言する場合、Amazon S3 バケットを作成、読み取り、更新、削除する権限が必要になります。ロールを必要最小限の権限に制限することが、セキュリティのベストプラクティスだと思われます。幅広い AWS リソースを考慮すると、特にそれらのテンプレートによって管理されているリソースが後で変更される可能性がある場合、IaaC テンプレートの最小特権ポリシーを作成することは困難です。例えば、管理対象のテンプレートに対する最新の編集では AWS Proton、RDS データベースリソースを追加します。

適切なアクセス許可を設定すると、IaC のデプロイをスムーズに行うことができます。 AWS Proton CodeBuild プロビジョニングは、お客様のアカウントにある CodeBuild プロジェクトで、お客様が提供した任意の CLI コマンドを実行します。通常、これらのコマンドは、 AWS CDKなどのコードとしてのインフラストラクチャ (IaaC) ツールを使用してインフラストラクチャの作成と削除を行います。テンプレートが CodeBuild プロビジョニングを使用する AWS リソースがデプロイされると、 AWS は によって管理される CodeBuild プロジェクトでビルドを開始します AWS。ロールが CodeBuild に渡され、CodeBuild はコマンドの実行を引き受けます。このロールを CodeBuild プロビジョニングロールと呼び、提供するのはカスタマーであり、インフラストラクチャのプロビジョニングに必要な権限が含まれます。これは CodeBuild のみが引き受けるものであり、引き受け AWS Proton ることはできません。

ロールの作成

CodeBuild プロビジョニングロールは IAM コンソールや AWS CLIで作成できます。 AWS CLIで作成する:

```
aws iam create-role --role-name AWSProtonCodeBuildProvisioning --assume-role-policy-
document '{"Version": "2012-10-17",       "Statement":[{"Effect":"Allow","Principal":
{"Service":"codebuild.amazonaws.com"},"Action":"sts:AssumeRole"}]}'
aws iam attach-role-policy --role-name AWSProtonCodeBuildProvisioning --policy-arn
 arn:aws:iam::aws:policy/AWSProtonCodeBuildProvisioningBasicAccess
```

これにより、CodeBuild サービスがビルドを実行するのに必要な最小限の権限を含む `AWSProtonCodeBuildProvisioningBasicAccess` も一緒にアタッチされます。

コンソールを使用する場合は、ロールの作成時に以下の点を確認してください。

1. 信頼されたエンティティの場合は、 AWS サービスを選択し、CodeBuild を選択します。
2. 「権限の追加」ステップで、`AWSProtonCodeBuildProvisioningBasicAccess` を選択しアタッチしたいその他のポリシーを選択します。

管理者アクセス権

AdministratorAccess ポリシーを CodeBuild プロビジョニングロールにアタッチすると、権限不足による障害が IaaC テンプレートに発生しなくなります。また、環境テンプレートまたはサービステンプレートを作成できるユーザーは、そのユーザーが管理者でなくても、管理者レベルのアクションを実行できます。CodeBuild プロビジョニングロールAdministatorAccessで AWS Proton を使用することはお勧めしません。CodeBuild プロビジョニングロールと一緒に AdministratorAccess を使用する場合は、サンドボックス環境で使用してください。

IAM コンソールで、またはこのコマンドを実行することで、AdministratorAccess でロールを作成できます：

```
aws iam create-role --role-name AWSProtonCodeBuildProvisioning --assume-role-policy-
document '{"Version": "2012-10-17",       "Statement":[{"Effect":"Allow","Principal":
{"Service":"codebuild.amazonaws.com"},"Action":"sts:AssumeRole"}]}'
aws iam attach-role-policy --role-name AWSProtonCodeBuildProvisioning --policy-arn
 arn:aws:iam::aws:policy/AdministratorAccess
```

最小スコープのロールの作成

最小限の権限でロールを作成したい場合は、複数の方法があります。

- 管理者権限でデプロイし、ロールの範囲を絞り込みます。IAM Access Analyzer の使用をお勧めします。
- 管理ポリシーを使用して、使用する予定のサービスへのアクセスを許可します。

AWS CDK

AWS CDK で を使用していて AWS Proton、各環境アカウント/リージョンcdk bootstrapで を実行している場合、 のロールが既に存在しますcdk deploy。この場合、次のポリシーを CodeBuild プロビジョニングロールにアタッチします：

```
{
    "Action": "sts:AssumeRole",
    "Resource": [
        "arn:aws:iam::account-id:role/cdk-*-deploy-role-*",
        "arn:aws:iam::account-id:role/cdk-*-file-publishing-role-*"
    ],
    "Effect": "Allow"
}
```

カスタム VPC

CodeBuild をカスタム VPC で実行する場合は、あなたの CodeBuild ロールに次の権限が必要です。

```
{
    "Effect": "Allow",
    "Action": [
        "ec2:CreateNetworkInterface"
    ],
    "Resource": [
        "arn:aws:ec2:region:account-id:network-interface/*",
        "arn:aws:ec2:region:account-id:subnet/*",
        "arn:aws:ec2:region:account-id:security-group/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "ec2:DeleteNetworkInterface"
    ],
    "Resource": [
        "arn:aws:ec2:region:account-id:*/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeVpcs"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "ec2:CreateNetworkInterfacePermission"
    ],
    "Resource": "arn:aws:ec2:region:account-id:network-interface/*",
    "Condition": {
        "StringEquals": {
            "ec2:AuthorizedService": "codebuild.amazonaws.com"
        }
    }
```

```
}
```

AmazonEC2FullAccess 管理ポリシーを使用することもできますが、管理ポリシーには必要のない権限が含まれる場合があります。 CLI を使用してマネージドポリシーをアタッチするには、以下の手順に従います：

```
aws iam create-role --role-name AWSProtonCodeBuildProvisioning --assume-role-policy-
document '{"Version": "2012-10-17",       "Statement":[{"Effect":"Allow","Principal":
{"Service":"codebuild.amazonaws.com"},"Action":"sts:AssumeRole"}]}'
aws iam attach-role-policy --role-name AWSProtonCodeBuildProvisioning --policy-arn
 arn:aws:iam::aws:policy/AdministratorAccess
```

AWS Proton サービス

AWS Proton サービスはサービステンプレートのインスタンス化であり、通常は複数のサービスインスタンスとパイプラインが含まれます。 AWS Proton サービスインスタンスは、特定の環境におけるサービステンプレートのインスタンス化です。サービステンプレートは、 サービスのインフラストラクチャとオプションの AWS Proton サービスパイプラインの完全な定義です。

サービスインスタンスをデプロイした後、CI/CD パイプラインをプロンプトするソースコードプッシュ、またはサービステンプレートの新しいバージョンにサービスを更新することで、インスタンスを更新できます。 は、サービステンプレートの新しいバージョンが利用可能になったときに AWS Proton プロンプトを表示し、サービスを新しいバージョンに更新できるようにします。サービスが更新されると、 はサービスとサービスインスタンス AWS Proton を再デプロイします。

この章では、作成、表示、更新、および削除の操作を通してサービスの管理方法を説明します。詳細については、「 AWS Proton サービス API リファレンス」を参照してください。

トピック

サービスを作成する

開発者として を使用してアプリケーションをデプロイするには AWS Proton、サービスを作成し、次の入力を指定します。

1. プラットフォームチームが発行する AWS Proton サービステンプレートの名前。
2. サービスの名前。
3. デプロイする必要があるサービスインスタンスの数。
4. 使用したい環境の選択。

5. サービスパイプラインを含むサービステンプレートを使用している場合、コードリポジトリへの接続 (オプション)。

サービスには何が含まれていますか?

AWS Proton サービスを作成するときは、2 つの異なるタイプのサービステンプレートから選択できます。

- サービスパイプラインを含むサービステンプレート (デフォルト)。
- サービスパイプラインを含まないサービステンプレート (デフォルト)。

サービスを作成する際には 1 つのサービスインスタンスを作成する必要があります。

サービスインスタンスとオプションのパイプラインはサービスに関連付けられています。パイプラインインスタンスを作成または削除できるのは、サービスの作成および削除アクションのコンテキスト内のみに限られます。サービスからインスタンスを追加および削除する方法については、「サービスを編集する」を参照してください。

> **Note**
>
> 環境は、環境で使用するのと同じプロビジョニング方法を使用して、環境内の AWS Proton サービス AWSまたはセルフマネージドプロビジョニング用に設定されています。サービスインスタンスの作成や更新をする開発者は設定の違いを認識できず、どちらの場合も行うことは同じです。
> プロビジョニング方法については、「the section called “プロビジョニングの方法”」を参照してください。

サービステンプレート

サービステンプレートのメジャーバージョンとマイナーバージョンの両方を使用できます。コンソールを使用する際には、サービステンプレートの最新の Recommended メジャーバージョンとマイナーバージョンを選択します。を使用し AWS CLI 、サービステンプレートのメジャーバージョンのみを指定する場合は、最新のRecommendedマイナーバージョンを暗黙的に指定します。

以下では、メジャーとマイナーのテンプレートバージョンの違いと、その使用方法について説明します。

- テンプレートの新しいバージョンは、プラットフォームチームのメンバーによって承認されるとすぐに Recommended になります。つまり、そのバージョンを使用して新しいサービスが作成され、既存のサービスを新しいバージョンに更新するように求められます。
- を通じて AWS Proton、プラットフォームチームはサービスインスタンスを新しいマイナーバージョンのサービステンプレートに自動的に更新できます。マイナーバージョンは下位互換でなければなりません。
- メジャーバージョンでは、更新プロセスの一環として新しい入力を提供する必要があるので、サービスをそのサービステンプレートのメジャーバージョンに更新する必要があります。メジャーバージョンは下位互換ではありません。

[Create a service (サービスを作成)]

次の手順は、 AWS Proton コンソールまたは を使用して AWS CLI 、サービスパイプラインの有無にかかわらずサービスを作成する方法を示しています。

AWS マネジメントコンソール

以下の手順に示すように、コンソールで標準サービスを作成します。

1. AWS Proton コンソールで、[Services (サービス)] を選択します。
2. [Create service (サービスの作成)] を選択します。
3. [Choose a service template (サービステンプレートを選択する)] ページでテンプレートを選択して [Configure (設定する)] を選択します。

   有効化したパイプラインを使用したくない場合、サービスについて [Excludes pipeline (パイプラインを除外)] のマークが付いたテンプレートを選択します。
4. [Configure service (サービスを構成する)] ページで [Service settings (サービス設定)] セクションの [Service name] にサービス名を入力します。
5. (オプション) サービスの説明を入力します。
6. [Service repository settings (サービスリポジトリ設定)] セクションで、次のように操作します。

   a. [CodeStar Connection (CodeStar 接続)] でリストから接続を選択します。

   b. [Repository ID (リポジトリ ID)] でソースコードが含まれているリポジトリの名前を選択します。

c. [Branch name (ブランチ名)] でソースコードが含まれているリポジトリの名前を選択します。

7. (オプション) [Tags (タグ)] セクションで [Add new tag (新しいタグを追加)] を選択し、キーと値を入力してカスタマーマネージドタグを作成します。
8. [Next (次へ)] を選択します。
9. [Configure custom settings (カスタム設定の構成)] ページの [Service instaces (サービスインスタンス)] セクションで [New instance (新しいインスタンス)] を選択します。`required` パラメータの値を入力する必要があります。`optional` パラメータの値を入力するか、表示されるデフォルト値を使用します。
10. [Pipeline inputs (パイプライン入力)] セクションで `required` パラメータの値を入力する必要があります。`optional` パラメータの値を入力するか、表示されるデフォルト値を使用します。
11. [Next (次へ)] を選択して入力を見直します。
12. [作成] を選択します。

    サービスの詳細とステータス、サービスの AWS マネージドタグとカスタマーマネージドタグを表示します。
13. ナビゲーションペインで [Services (サービス)] を選択します。

    新しいページには、ステータスやその他のサービスの詳細とともに、サービスのリストが表示されます。

AWS CLI

を使用する場合は AWS CLI、ソースコードディレクトリにある YAML 形式のspecファイル `.aws-proton/service.yaml`でサービス入力を指定します。

CLI で `get-service-template-minor-version` コマンドを使用して、仕様ファイル内で値を指定するスキーマの必須パラメータとオプションパラメータを表示します。

`pipelineProvisioning: "CUSTOMER_MANAGED"` を含むサービステンプレートを使用する場合、仕様に `pipeline:` セクションを含めないことと `create-service` コマンドに `-repository-connection-arn`、`-repository-id`、および `-branch-name` パラメータを含めないことが条件です。

次の手順に示すように、CLI でサービスパイプライン付きのサービスを作成します。

1. 次の CLI コマンドの例に示すように、パイプラインのサービスロールをセットアップします。

   コマンド:

   ```
   $ aws proton update-account-settings \
           --pipeline-service-role-arn "arn:aws:iam::123456789012:role/AWS
    ProtonServiceRole"
   ```

2. 以下のリストは、サービステンプレートスキーマに基づいて、サービスパイプラインとインスタンスの入力を含めた仕様の例を示しています。

   仕様:

   ```
   proton: ServiceSpec

   pipeline:
     my_sample_pipeline_required_input: "hello"
     my_sample_pipeline_optional_input: "bye"

   instances:
     - name: "acme-network-dev"
       environment: "ENV_NAME"
       spec:
         my_sample_service_instance_required_input: "hi"
         my_sample_service_instance_optional_input: "ho"
   ```

   次の CLI コマンドとレスポンスの例に示すように、パイプライン付きでサービスを作成します。

   コマンド:

   ```
   $ aws proton create-service \
           --name "MySimpleService" \
           --branch-name "mainline" \
           --template-major-version "1" \
           --template-name "fargate-service" \
           --repository-connection-arn "arn:aws:codestar-connections:region-
   id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111" \
           --repository-id "myorg/myapp" \
           --spec "file://spec.yaml"
   ```

レスポンス:

```
{
    "service": {
        "arn": "arn:aws:proton:region-id:123456789012:service/MySimpleService",
        "createdAt": "2020-11-18T19:50:27.460000+00:00",
        "lastModifiedAt": "2020-11-18T19:50:27.460000+00:00",
        "name": "MySimpleService",
        "repositoryConnectionArn": "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "repositoryId": "myorg/myapp",
        "status": "CREATE_IN_PROGRESS",
        "templateName": "fargate-service"
    }
}
```

次の CLI コマンドとレスポンスの例に示すように、サービスパイプラインなしでサービスを作成します。

以下に示すのは、サービスパイプライン入力を含めない仕様の例です。

仕様:

```
proton: ServiceSpec

instances:
  - name: "acme-network-dev"
    environment: "ENV_NAME"
    spec:
      my_sample_service_instance_required_input: "hi"
      my_sample_service_instance_optional_input: "ho"
```

プロビジョニングされたサービスパイプラインなしでサービスを作成するには、次の CLI コマンドとレスポンスの例に示すように、**spec.yaml** へのパスを指定し、かつリポジトリパラメータを含めないことです。

コマンド:

```
$ aws proton create-service \
        --name "MySimpleServiceNoPipeline" \
```

```
        --template-major-version "1" \
        --template-name "fargate-service" \
        --spec "file://spec-no-pipeline.yaml"
```

レスポンス:

```
{
    "service": {
        "arn": "arn:aws:proton:region-id:123456789012:service/
MySimpleServiceNoPipeline",
        "createdAt": "2020-11-18T19:50:27.460000+00:00",
        "lastModifiedAt": "2020-11-18T19:50:27.460000+00:00",
        "name": "MySimpleServiceNoPipeline",
        "status": "CREATE_IN_PROGRESS",
        "templateName": "fargate-service-no-pipeline"
    }
}
```

サービスデータを表示する

AWS Proton コンソールまたは を使用して、サービスの詳細データを表示および一覧表示できます AWS CLI。

AWS マネジメントコンソール

次の手順に示すように、AWS Proton コンソールで、サービスを一覧表示して詳細を確認します。

1. サービスの一覧を表示するには、ナビゲーションペインで [Services (サービス)] を選択します。
2. 詳細データを表示するには、サービスの名前を選択します。

   サービスの詳細データを表示します。

AWS CLI

次の CLI のコマンドとレスポンスの例に示すように、サービスパイプライン付きのサービスの詳細を表示します。

コマンド:

```
$ aws proton get-service \
    --name "simple-svc"
```

レスポンス:

```
{
    "service": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc",
        "branchName": "mainline",
        "createdAt": "2020-11-28T22:40:50.512000+00:00",
        "lastModifiedAt": "2020-11-28T22:44:51.207000+00:00",
        "name": "simple-svc",
        "pipeline": {
            "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/
pipeline/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
            "createdAt": "2020-11-28T22:40:50.512000+00:00",
            "deploymentStatus": "SUCCEEDED",
            "lastDeploymentAttemptedAt": "2020-11-28T22:40:50.512000+00:00",
            "lastDeploymentSucceededAt": "2020-11-28T22:40:50.512000+00:00",
            "spec": "proton: ServiceSpec\npipeline:\n
 my_sample_pipeline_required_input: hello\n  my_sample_pipeline_optional_input:
 bye\ninstances:\n- name: instance-svc-simple\n  environment: my-simple-
env\n  spec:\n    my_sample_service_instance_required_input: hi\n
 my_sample_service_instance_optional_input: ho\n",
            "templateMajorVersion": "1",
            "templateMinorVersion": "1",
            "templateName": "svc-simple"
        },
        "repositoryConnectionArn": "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
        "repositoryId": "myorg/myapp",
        "spec": "proton: ServiceSpec\npipeline:\n
 my_sample_pipeline_required_input: hello\n  my_sample_pipeline_optional_input:
 bye\ninstances:\n- name: instance-svc-simple\n  environment: my-simple-
env\n  spec:\n    my_sample_service_instance_required_input: hi\n
 my_sample_service_instance_optional_input: ho\n",
        "status": "ACTIVE",
        "templateName": "svc-simple"
    }
}
```

次の CLI のコマンドとレスポンスの例に示すように、サービスパイプラインなしのサービスの詳細を表示します。

コマンド:

```
$ aws proton get-service \
    --name "simple-svc-no-pipeline"
```

レスポンス:

```
{
    "service": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc-without-
pipeline",
        "createdAt": "2020-11-28T22:40:50.512000+00:00",
        "lastModifiedAt": "2020-11-28T22:44:51.207000+00:00",
        "name": "simple-svc-without-pipeline",
        "spec": "proton: ServiceSpec\ninstances:\n- name: instance-svc-simple\n
 environment: my-simple-env\n  spec:\n    my_sample_service_instance_required_input:
 hi\n    my_sample_service_instance_optional_input: ho\n",
        "status": "ACTIVE",
        "templateName": "svc-simple-no-pipeline"
    }
}
```

サービスを編集する

AWS Proton サービスに対して以下の編集を行うことができます。

- サービスの説明を編集する
- サービスインスタンスを追加や削除することでサービスを編集します。

サービスの説明を編集する

コンソールまたは を使用して AWS CLI 、サービスの説明を編集できます。

AWS マネジメントコンソール

以下で説明する手順に従って、コンソールでサービスを編集します。

サービスのリストで次のように操作します。

1. AWS Proton コンソールで、[Services (サービス)] を選択します。
2. サービスの一覧で、更新したいサービスの左側にあるラジオボタンを選択します。
3. [Edit] (編集) を選択します。
4. [Configure service (サービスを構成する)] ページでフォームに値を入力して [Next (次へ)] を選択します。
5. [Configure custom settings (カスタム設定の構成)] ページで [Next (次へ)] を選択します。
6. 編集内容を見直して [[Save changes (変更を保存)] を選択します。

サービスの詳細ページで、以下の操作をします。

1. AWS Proton コンソールで、[Services (サービス)] を選択します。
2. サービスの一覧で、編集したいサービスの名前を選択します。
3. サービスの詳細ページで [Edit (編集)] を選択します。
4. [Configure service (サービスを構成する)] ページでフォームに値を入力して [Next (次へ)] を選択します。
5. [Configure custom settings (カスタム設定を構成する)] ページでフォームに値を入力して [Next (次へ)] を選択します。
6. 編集内容を見直して [Save changes (変更を保存)] を選択します。

AWS CLI

次の CLI の例に示すように説明を編集します。

コマンド:

```
$ aws proton update-service \
    --name "MySimpleService" \
    --description "Edit by updating description"
```

レスポンス:

```
{
    "service": {
        "arn": "arn:aws:proton:region-id:123456789012:service/MySimpleService",
```

```
        "branchName": "main",
        "createdAt": "2021-03-12T22:39:42.318000+00:00",
        "description": "Edit by updating description",
        "lastModifiedAt": "2021-03-12T22:44:21.975000+00:00",
        "name": "MySimpleService",
        "repositoryConnectionArn": "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "repositoryId": "my-repository/myorg-myapp",
        "status": "ACTIVE",
        "templateName": "fargate-service"
    }
}
```

サービスインスタンスの追加や削除によってサービスを編集します。

AWS Proton サービスの場合は、編集した仕様を送信することで、サービスインスタンスを追加または削除できます。リクエストを成功させるためには、次の条件が満たされている必要があります。

- 編集リクエストの送信時にサービスとパイプラインの編集や削除が行われていない。
- 編集した仕様にサービスパイプラインや削除対象ではない既存のサービスインスタンスを変更する編集が行われていない。
- 編集した仕様で、コンポーネントがアタッチされている既存のサービスインスタンスが削除されていない。このようなサービスインスタンスを削除するには、まずコンポーネントを更新してサービスインスタンスからデタッチする必要があります。コンポーネントの詳細については、「コンポーネント」を参照してください。

削除に失敗したインスタンスは、DELETE_FAILED 状態のサービスインすんタンスです。サービスの編集をリクエストすると、 は編集プロセスの一環として、削除に失敗したインスタンスを自動的に削除 AWS Proton しようとします。いずれかのサービスインスタンスの削除に失敗した場合、コンソールまたは AWS CLIに表示されなくても、インスタンスに関連付けられたリソースがまだ残っている可能性があります。削除に失敗したインスタンスインフラストラクチャリソースを確認し、AWS Proton が自動的に削除できるようにクリーンアップします。

サービスのサービスインスタンスのクォータについては、「AWS Proton クォータ」を参照してください。また、サービスの作成後に 1 つ以上のサービスインスタンスを維持する必要があります。更新プロセス中に、 は既存のサービスインスタンスと追加または削除されるインスタンスの数 AWS Proton を作成します。削除に失敗したインスタンスもこの数に含まれるため、spec の編集時に考慮に入れる必要があります。

コンソールまたは AWS CLI を使用して、サービスインスタンスを追加または削除します。

AWS マネジメントコンソール

コンソールでサービスインスタンスを追加するか削除してサービスを編集します。

AWS Proton コンソール内

1. ナビゲーションペインで [Services (サービス)] を選択します。
2. 編集するサービスを選択します。
3. [Edit (編集)] を選択します。
4. (オプション) [Configure service (サービスを構成する)] ページで、サービス名または説明を編集して、ページの右下にある [Next (次へ)] を選択します。
5. [Configure custom settings (カスタム設定の構成)] ページで [Delete (削除)] を選択してサービスインスタンスを削除し、[Add new instance (新しいインスタンスを追加)] を選択してサービスインスタンスを追加し、フォームに値を入力します。
6. [Next (次へ)] をクリックします。
7. 更新内容を確認して [Save changes (変更を保存)] を選択します。
8. モーダルがサービスインスタンスの削除の確認を求めるメッセージを表示します。指示に従って操作し、[Yes, delete (はい、削除します)] を選択します。
9. サービスの詳細ページに表示されるサービスのステータスを確認します。

AWS CLI

次のコマンドとレスポンス AWS CLI の例**spec**に示すように、 を編集してサービスインスタンスを追加および削除します。

CLI を使用する場合、`spec` で削除するサービスインスタンスを除外し、追加するサービスインスタンスと削除マークを付けていない既存のサービスインスタンスの両方を含める必要があります。

以下のリストは、編集前の `spec` と仕様によってデプロイされるサービスインスタンスのリストの例を示します。この仕様は、サービスの説明を編集するために前の例で使用されたものです。

仕様:

```
proton: ServiceSpec

pipeline:
  my_sample_pipeline_optional_input: "abc"
  my_sample_pipeline_required_input: "123"

instances:
  - name: "my-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_optional_input: "def"
      my_sample_service_instance_required_input: "456"
  - name: "my-other-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_required_input: "789"
```

次の CLI `list-service-instances` コマンドとレスポンスの例は、サービスインスタンスを追加または削除する前にアクティブなインスタンスを表示します。

コマンド:

```
$ aws proton list-service-instances \
    --service-name "MySimpleService"
```

レスポンス:

```
{
    "serviceInstances": [
        {
            "arn": "arn:aws:proton:region-id:123456789012:service/MySimpleService/
service-instance/my-other-instance",
            "createdAt": "2021-03-12T22:39:42.318000+00:00",
            "deploymentStatus": "SUCCEEDED",
            "environmentName": "simple-env",
            "lastDeploymentAttemptedAt": "2021-03-12T22:39:43.109000+00:00",
            "lastDeploymentSucceededAt": "2021-03-12T22:39:43.109000+00:00",
            "name": "my-other-instance",
            "serviceName": "example-svc",
            "templateMajorVersion": "1",
            "templateMinorVersion": "0",
            "templateName": "fargate-service"
```

```
        },
        {
            "arn": "arn:aws:proton:region-id:123456789012:service/MySimpleService/
service-instance/my-instance",
            "createdAt": "2021-03-12T22:39:42.318000+00:00",
            "deploymentStatus": "SUCCEEDED",
            "environmentName": "simple-env",
            "lastDeploymentAttemptedAt": "2021-03-12T22:39:43.160000+00:00",
            "lastDeploymentSucceededAt": "2021-03-12T22:39:43.160000+00:00",
            "name": "my-instance",
            "serviceName": "example-svc",
            "serviceTemplateArn": "arn:aws:proton:region-id:123456789012:service-
template/fargate-service",
            "templateMajorVersion": "1",
            "templateMinorVersion": "0",
            "templateName": "fargate-service"
        }
    ]
}
```

次のリストは、インスタンスの削除や追加に編集した spec を使用する例を示します。my-instance という名前の既存のインスタンスが削除され、yet-another-instance という名前の新しいインスタンスが作成されます。

仕様:

```
proton: ServiceSpec

pipeline:
  my_sample_pipeline_optional_input: "abc"
  my_sample_pipeline_required_input: "123"

instances:
  - name: "my-other-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_required_input: "789"
  - name: "yet-another-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_required_input: "789"
```

"${Proton::CURRENT_VAL}" を使用すると、spec に値が存在する場合に元の spec を保持するパラメータ値を指定できます。サービスデータを表示する で説明するように、get-service を使用してサービスの元の spec を表示します。

以下のリストは、spec で既存のサービスインスタンスを確実に残すようにパラメータ値の変更を含めないで "${Proton::CURRENT_VAL}" を使用する方法を示します。

仕様:

```
proton: ServiceSpec

pipeline:
  my_sample_pipeline_optional_input: "${Proton::CURRENT_VAL}"
  my_sample_pipeline_required_input: "${Proton::CURRENT_VAL}"

instances:
  - name: "my-other-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_required_input: "${Proton::CURRENT_VAL}"
  - name: "yet-another-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_required_input: "789"
```

以下に示すのは、サービスを編集するための CLI コマンドとリクエストの一覧です。

コマンド:

```
$ aws proton update-service
    --name "MySimpleService" \
    --description "Edit by adding and deleting a service instance" \
    --spec "file://spec.yaml"
```

レスポンス:

```
{
    "service": {
        "arn": "arn:aws:proton:region-id:123456789012:service/MySimpleService",
        "branchName": "main",
        "createdAt": "2021-03-12T22:39:42.318000+00:00",
        "description": "Edit by adding and deleting a service instance",
```

```
        "lastModifiedAt": "2021-03-12T22:55:48.169000+00:00",
        "name": "MySimpleService",
        "repositoryConnectionArn": "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "repositoryId": "my-repository/myorg-myapp",
        "status": "UPDATE_IN_PROGRESS",
        "templateName": "fargate-service"
    }
}
```

次の list-service-instances コマンドとレスポンスは、my-instance という名前の既存のインスタンスが削除され、yet-another-instance という名前の新しいインスタンスが作成されたことを確認します。

コマンド:

```
$ aws proton list-service-instances \
    --service-name "MySimpleService"
```

レスポンス:

```
{
    "serviceInstances": [
        {
            "arn": "arn:aws:proton:region-id:123456789012:service/MySimpleService/
service-instance/yet-another-instance",
            "createdAt": "2021-03-12T22:39:42.318000+00:00",
            "deploymentStatus": "SUCCEEDED",
            "environmentName": "simple-env",
            "lastDeploymentAttemptedAt": "2021-03-12T22:56:01.565000+00:00",
            "lastDeploymentSucceededAt": "2021-03-12T22:56:01.565000+00:00",
            "name": "yet-another-instance",
            "serviceName": "MySimpleService",
            "templateMajorVersion": "1",
            "templateMinorVersion": "0",
            "templateName": "fargate-service"
        },
        {
            "arn": "arn:aws:proton:region-id:123456789012:service/MySimpleService/
service-instance/my-other-instance",
            "createdAt": "2021-03-12T22:39:42.318000+00:00",
            "deploymentStatus": "SUCCEEDED",
            "environmentName": "simple-env",
```

```
            "lastDeploymentAttemptedAt": "2021-03-12T22:39:43.109000+00:00",
            "lastDeploymentSucceededAt": "2021-03-12T22:39:43.109000+00:00",
            "name": "my-other-instance",
            "serviceName": "MySimpleService",
            "templateMajorVersion": "1",
            "templateMinorVersion": "0",
            "templateName": "fargate-service"
        }
    ]
}
```

サービスインスタンスを追加または削除した場合

サービス編集を送信してサービスインスタンスを削除および追加すると、 AWS Proton は次のアクションを実行します。

- サービスを UPDATE_IN_PROGRESS に設定します。
- サービスにパイプラインがある場合、そのステータスを IN_PROGRESS に設定してパイプラインアクションをブロックします。
- 削除したいサービスインスタンスを DELETE_IN_PROGRESS に設定します。
- サービスアクションをブロックします。
- 削除対象のマークが付いたサービスインスタンスに対するアクションをブロックします。
- 新しいサービスインスタンスを作成します。
- 削除対象のリストにあるインスタンスを削除します。
- 削除に失敗したインスタンスの削除を試みます。
- 追加と削除が完了したら、サービスパイプライン (存在する場合) を再プロビジョニングし、サービスを ACTIVE に設定してサービスとパイプラインのアクションを有効にします。

AWS Proton は、次のように障害モードを修復しようとします。

- 1 つ以上のサービスインスタンスの作成に失敗した場合、 は新しく作成されたすべてのサービスインスタンスのプロビジョニングを解除 AWS Proton しようとし、 specを以前の状態に戻します。どのサービスインスタンスも削除されず、パイプラインはいかなる修正もされません。
- 1 つ以上のサービスインスタンスを削除できなかった場合、 は削除されたインスタンスなしでパイプライン AWS Proton を再プロビジョニングします。spec は、追加されたインスタンスを含め、削除マークの付いたインスタンスを除外するように更新されます。

- パイプラインのプロビジョニングに失敗した場合、ロールバックの試みはなされず、サービスとパイプラインの両方が失敗した更新状態を反映します。

タグ付けとサービスの編集

サービス編集の一部としてサービスインスタンスを追加すると、 AWS マネージドタグが に伝播され、新しいインスタンスとプロビジョニングされたリソースに対して自動的に作成されます。新しいタグを作成すると、それらのタグは新しいインスタンスにのみ適用されます。既存のサービスカスタマーマネージドタグも新しいインスタンスに伝播します。詳細については、「AWS Proton リソースとタグ付け」を参照してください。

サービスを削除する

AWS Proton コンソールまたは を使用して、インスタンスとパイプラインを含む AWS Proton サービスを削除できます AWS CLI。

コンポーネントがアタッチされたサービスインスタンスがあるサービスは削除できません。このようなサービスを削除するには、まず、アタッチされているすべてのコンポーネントを更新して、サービスインスタンスからデタッチする必要があります。コンポーネントの詳細については、「コンポーネント」を参照してください。

AWS マネジメントコンソール

以下で説明する手順に従って、コンソールでサービスを削除します。

サービスの詳細ページで、以下の操作をします。

1. AWS Proton コンソールで、[Services (サービス)] を選択します。
2. サービスの一覧で、削除したいサービスの名前を選択します。
3. サービスの詳細ページで [Actions (アクション)] を選択してから [Delete (削除)] を選択します。
4. モーダルから Delete アクションの確認を求めるプロンプトが表示されます。
5. 指示に従って操作し、[Yes, delete (はい、削除します)] を選択します。

AWS CLI

次の CLI の例に示すようにサービスを削除します。

コマンド:

```
$ aws proton delete-service \
    --name "simple-svc"
```

レスポンス:

```
{
    "service": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc",
        "branchName": "mainline",
        "createdAt": "2020-11-28T22:40:50.512000+00:00",
        "description": "Edit by updating description",
        "lastModifiedAt": "2020-11-29T00:30:39.248000+00:00",
        "name": "simple-svc",
        "repositoryConnectionArn": "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "repositoryId": "myorg/myapp",
        "status": "DELETE_IN_PROGRESS",
        "templateName": "fargate-service"
    }
}
```

サービスインスタンスデータの表示

AWS Proton サービスインスタンスの詳細データを表示する方法について説明します。コンソールまたは AWS CLIを使用できます。

サービスインスタンスはサービスに属します。インスタンスを作成または削除できるのは、サービスの編集、作成、および削除アクションのコンテキスト内のみに限られます。サービスからインスタンスを追加および削除する方法については、「サービスを編集する」を参照してください。

AWS マネジメントコンソール

次の手順に示すように、AWS Proton コンソールで、サービスインスタンスを一覧表示して詳細を確認します。

1. サービスインスタンスの一覧を表示するには、ナビゲーションペインで [Service instances (サービスインスタンス)] 選択します。
2. 詳細データを表示するには、サービスインスタンスの名前を選択します。

サービスインスタンスの詳細データを表示します。

AWS CLI

次の CLI コマンドとレスポンスの例に示すように、サービスインスタンスの詳細な一覧を表示します。

コマンド:

```
$ aws proton list-service-instances
```

レスポンス:

```
{
    "serviceInstances": [
        {
            "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/
service-instance/instance-one",
            "createdAt": "2020-11-28T22:40:50.512000+00:00",
            "deploymentStatus": "SUCCEEDED",
            "environmentArn": "arn:aws:proton:region-id:123456789012:environment/
simple-env",
            "lastDeploymentAttemptedAt": "2020-11-28T22:40:50.512000+00:00",
            "lastDeploymentSucceededAt": "2020-11-28T22:40:50.512000+00:00",
            "name": "instance-one",
            "serviceName": "simple-svc",
            "templateMajorVersion": "1",
            "templateMinorVersion": "0",
            "templateName": "fargate-service"
        }
    ]
}
```

コマンド:

```
$ aws proton get-service-instance \
    --name "instance-one" \
    --service-name "simple-svc"
```

レスポンス:

```
{
    "serviceInstance": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/service-
instance/instance-one",
        "createdAt": "2020-11-28T22:40:50.512000+00:00",
        "deploymentStatus": "SUCCEEDED",
        "environmentName": "simple-env",
        "lastDeploymentAttemptedAt": "2020-11-28T22:40:50.512000+00:00",
        "lastDeploymentSucceededAt": "2020-11-28T22:40:50.512000+00:00",
        "name": "instance-one",
        "serviceName": "simple-svc",
        "spec": "proton: ServiceSpec\npipeline:\n
 my_sample_pipeline_optional_input: hello world\n
 my_sample_pipeline_required_input: pipeline up\ninstances:\n- name: instance-one\n
 environment: my-simple-env\n  spec:\n    my_sample_service_instance_optional_input:
 Ola\n    my_sample_service_instance_required_input: Ciao\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "0",
        "templateName": "svc-simple"
    }
}
```


サービスインスタンスを更新する

AWS Proton サービスインスタンスを更新し、更新をキャンセルする方法について説明します。

サービスインスタンスはサービスに属します。インスタンスを作成または削除できるのは、サービスの編集、作成、および削除アクションのコンテキスト内のみに限られます。サービスからインスタンスを追加および削除する方法については、「サービスを編集する」を参照してください。

サービスインスタンスを更新しようとする場合、以下に挙げるように 4 つのモードがあります。を使用する場合 AWS CLI、 `deployment-type`フィールドは モードを定義します。コンソールを使用する場合、これらのモードはサービスインスタンスの詳細ページにある [Actions (アクション)] ドロップダウンリストに表示される [Edit (編集)] と [Update to latest minor version (最新のマイナーバージョンに更新)] と [Update to latest major version (最新のメジャーバージョンに更新)] にマップされます。

`NONE`

このモードでは、デプロイは発生しません。リクエストしたメタデータパラメータのみが更新されます。

CURRENT_VERSION

このモードでは、サービスインスタンスがデプロイされ、指定した新しい仕様に従って更新されます。リクエストしたパラメータのみが更新されます。この deployment-type を使用する場合、マイナーバージョンまたはメジャーバージョンのパラメータを含めないでください。

MINOR_VERSION

このモードでは、サービスインスタンスは、デフォルトで、使用中の現行メジャーバージョンに所属する、パブリッシュされた推奨マイナーバージョン (最新) でデプロイされ、更新されます。また、使用中の現行メジャーバージョンの別のマイナーバージョンを指定することもできます。

MAJOR_VERSION

このモードでは、サービスインスタンスがデプロイされ、デフォルトで現在のテンプレートのパブリッシュされた推奨 (最新の) メジャーバージョンとマイナーバージョンで更新されます。使用中のメジャーバージョンやマイナーバージョン (オプション) よりも高いメジャーバージョンも指定できます。

deploymentStatus が IN_PROGRESS. AWS Proton attempts の場合、サービスインスタンスの更新デプロイをキャンセルできます。正常にキャンセルされる保証はありません。

更新デプロイをキャンセルすると、 は次のステップに示すようにデプロイをキャンセル AWS Proton しようとします。

- デプロイの状態を「CANCELLING」に設定します。
- 処理中のデプロイを停止し、IN_PROGRESS の際にデプロイによって作成された新しいリソースを削除します。
- デプロイの状態を「CANCELLED」に設定します。
- リソースの状態をデプロイが開始される前の状態に戻します。

サービスインスタンスのデプロイをキャンセルする方法の詳細については、『AWS Proton API リファレンス』の「CancelServiceInstanceDeployment」を参照してください。

コンソールまたは AWS CLI を使用して、更新を行うか、更新デプロイをキャンセルします。

AWS マネジメントコンソール

次の手順に従って、コンソールでサービスインスタンスを更新します。

1. AWS Proton コンソールのサービスナビゲーションペインで [Service instance (サービスインスタンス)] を選択します。
2. サービスインスタンスの一覧で、更新したいサービスインスタンスの名前を選択します。
3. 仕様を更新するには [Actions (アクション)] を選択してから更新オプションのいずれかを選択して [Edit (編集)] で仕様を更新するか、または [Actions (アクション)] を選択してから [Update to latest minor version (最新のマイナーバージョンに更新)] または [Update to latest major version (最新のメジャーバージョンに更新)] を選択します。
4. 各フォームに必要な値を入力し、[Review (確認)] ページが表示されるまでに [Next (次へ)] を選択します。
5. 編集内容を確認して [Update (更新)] を選択します。

AWS CLI

CLI のコマンドとレスポンスの例に示すように、サービスインスタンスを更新して新しいマイナーバージョンにします。

修正した `spec` でサービスインスタンスを更新する際には、`"${Proton::CURRENT_VAL}"` を使用して `spec` に値が存在する場合に元の `spec` を保存するパラメータ値を指定できます。`get-service` で説明するように、`spec` を使用してサービスインスタンスの元の サービスデータを表示する を表示します。

以下の例は、`"${Proton::CURRENT_VAL}"` での `spec` の使用方法を示します。

仕様:

```
proton: ServiceSpec

pipeline:
  my_sample_pipeline_optional_input: "${Proton::CURRENT_VAL}"
  my_sample_pipeline_required_input: "${Proton::CURRENT_VAL}"

instances:
  - name: "my-instance"
```

```
    environment: "simple-env"
    spec:
      my_sample_service_instance_optional_input: "${Proton::CURRENT_VAL}"
      my_sample_service_instance_required_input: "${Proton::CURRENT_VAL}"
  - name: "my-other-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_required_input: "789"
```

コマンド: 更新する

```
$ aws proton update-service-instance \
    --name "instance-one" \
    --service-name "simple-svc" \
    --spec "file://service-spec.yaml" \
    --template-major-version "1" \
    --template-minor-version "1" \
    --deployment-type "MINOR_VERSION"
```

レスポンス:

```
{
    "serviceInstance": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/service-
instance/instance-one",
        "createdAt": "2021-04-02T21:29:59.962000+00:00",
        "deploymentStatus": "IN_PROGRESS",
        "environmentName": "arn:aws:proton:region-id:123456789012:environment/
simple-env",
        "lastDeploymentAttemptedAt": "2021-04-02T21:38:00.823000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T21:29:59.962000+00:00",
        "name": "instance-one",
        "serviceName": "simple-svc",
        "templateMajorVersion": "1",
        "templateMinorVersion": "0",
        "templateName": "svc-simple"
    }
}
```

コマンド: ステータスを取得して確認する

```
$ aws proton get-service-instance \
```

```
    --name "instance-one" \
    --service-name "simple-svc"
```

レスポンス:

```
{
    "serviceInstance": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/service-
instance/instance-one",
        "createdAt": "2021-04-02T21:29:59.962000+00:00",
        "deploymentStatus": "SUCCEEDED",
        "environmentName": "simple-env",
        "lastDeploymentAttemptedAt": "2021-04-02T21:38:00.823000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T21:38:00.823000+00:00",
        "name": "instance-one",
        "serviceName": "simple-svc",
        "spec": "proton: ServiceSpec\n\npipeline:\n
 my_sample_pipeline_optional_input: \"abc\"\n  my_sample_pipeline_required_input:
 \"123\"\n\ninstances:\n  - name: \"instance-one\"\n    environment: \"simple-
env\"\n    spec:\n      my_sample_service_instance_optional_input: \"def\"\n
      my_sample_service_instance_required_input: \"456\"\n  - name: \"my-
other-instance\"\n    environment: \"kls-simple-env\"\n    spec:\n
 my_sample_service_instance_required_input: \"789\"\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "svc-simple"
    }
}
```

AWS マネジメントコンソール

次の手順に示すように、コンソールでサービスインスタンスのデプロイをキャンセルします。

1. AWS Proton コンソールのサービスナビゲーションペインで [Service instance (サービスインスタンス)] を選択します。
2. サービスインスタンスの一覧で、デプロイ更新をキャンセルしたいサービスインスタンスの名前を選択します。
3. 更新のデプロイステータスが [In progress (進行中)] の場合、サービスインスタンスの詳細ページで [Action (アクション)] を選択してから [Cancel deployment (デプロイをキャンセル)] を選択します。

4. キャンセルを確認するメッセージが表示されます。[Cancel deployment (デプロイをキャンセル)] を選択します。
5. 更新のデプロイステータスが [[Cancelling (キャンセル中)] に変わり、キャンセルが完了すると [Cancelled (キャンセル済み)] に変わります。

AWS CLI

CLI のコマンドとレスポンスの例に示すように、新しいマイナーバージョン 2 への IN_PROGRESS サービスインスタンス更新をキャンセルします。

この例で使用するテンプレートには待機条件が含まれているため、更新のデプロイが成功する前にキャンセルが開始されます。

コマンド: キャンセルする

```
$ aws proton cancel-service-instance-deployment \
    --service-instance-name "instance-one" \
    --service-name "simple-svc"
```

レスポンス:

```
{
    "serviceInstance": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/service-
instance/instance-one",
        "createdAt": "2021-04-02T21:29:59.962000+00:00",
        "deploymentStatus": "CANCELLING",
        "environmentName": "simple-env",
        "lastDeploymentAttemptedAt": "2021-04-02T21:45:15.406000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T21:38:00.823000+00:00",
        "name": "instance-one",
        "serviceName": "simple-svc",
        "spec": "proton: ServiceSpec\npipeline:\n
 my_sample_pipeline_optional_input: abc\n  my_sample_pipeline_required_input:
 '123'\ninstances:\n- name: my-instance\n  environment: MySimpleEnv
\n  spec:\n    my_sample_service_instance_optional_input: def\n
 my_sample_service_instance_required_input: '456'\n- name: my-other-instance\n
 environment: MySimpleEnv\n  spec:\n    my_sample_service_instance_required_input:
 '789'\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
```

```
        "templateName": "svc-simple"
    }
}
```

コマンド: ステータスを取得して確認する

```
$ aws proton get-service-instance \
    --name "instance-one" \
    --service-name "simple-svc"
```

レスポンス:

```
{
    "serviceInstance": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/service-
instance/instance-one",
        "createdAt": "2021-04-02T21:29:59.962000+00:00",
        "deploymentStatus": "CANCELLED",
        "deploymentStatusMessage": "User initiated cancellation.",
        "environmentName": "simple-env",
        "lastDeploymentAttemptedAt": "2021-04-02T21:45:15.406000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T21:38:00.823000+00:00",
        "name": "instance-one",
        "serviceName": "simple-svc",
        "spec": "proton: ServiceSpec\n\npipeline:\n
 my_sample_pipeline_optional_input: \"abc\"\n  my_sample_pipeline_required_input:
 \"123\"\n\ninstances:\n  - name: \"instance-one\"\n    environment: \"simple-
env\"\n    spec:\n      my_sample_service_instance_optional_input: \"def\"\n
      my_sample_service_instance_required_input: \"456\"\n  - name: \"my-
other-instance\"\n    environment: \"kls-simple-env\"\n    spec:\n
 my_sample_service_instance_required_input: \"789\"\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "svc-simple"
    }
}
```

サービスパイプラインを更新する

AWS Proton サービスパイプラインを更新し、更新をキャンセルする方法について説明します。

サービスパイプラインはサービスに属します。パイプラインインスタンスを作成または削除できるのは、サービスの作成および削除アクションのコンテキスト内のみに限られます。

サービスパイプラインを更新する場合、以下の 4 つのモードがあります。を使用する場合 AWS CLI、 `deployment-type`フィールドは モードを定義します。コンソールを使用する場合、これらのモードは [Edit pipeline (パイプラインの編集)] と [Update to recommended version (推奨バージョンに更新)] にマップされます。

`NONE`

このモードでは、デプロイは発生しません。リクエストしたメタデータパラメータのみが更新されます。

`CURRENT_VERSION`

このモードでは、サービスパイプラインがデプロイされ、指定した新しい仕様に従って更新されます。リクエストしたパラメータのみが更新されます。この `deployment-type` を使用する場合、マイナーバージョンまたはメジャーバージョンのパラメータを含めないでください。

`MINOR_VERSION`

このモードでは、デフォルトで、サービスパイプラインは、使用中の現行メジャーバージョンのパブリッシュされた推奨 (最新の) マイナーバージョンでデプロイされ、更新されます。また、使用中の現在のメジャーバージョンについて別のマイナーバージョンを指定することもできます。

`MAJOR_VERSION`

このモードでは、デフォルトで、サービスパイプラインは、パブリッシュされた推奨 (最新の) メジャーバージョンとマイナーバージョンでデプロイされ、更新されます。使用中のメジャーバージョンやマイナーバージョン (オプション) よりも高いメジャーバージョンも指定できます。

`deploymentStatus` が `IN_PROGRESS`. AWS Proton attempts の場合、サービスパイプラインの更新デプロイをキャンセルしようとできます。正常にキャンセルされる保証はありません。

更新デプロイをキャンセルすると、 は次のステップに示すようにデプロイをキャンセル AWS Proton しようとします。

- デプロイの状態を「`CANCELLING`」に設定します。

- 処理中のデプロイを停止し、IN_PROGRESS の際にデプロイによって作成された新しいリソースを削除します。
- デプロイの状態を「CANCELLED」に設定します。
- リソースの状態をデプロイが開始される前の状態に戻します。

サービスパイプラインのデプロイをキャンセルする方法の詳細については、『AWS Proton API リファレンス』の「CancelServicePipelineDeployment」を参照してください。

コンソールまたは AWS CLI を使用して、更新を行うか、更新デプロイをキャンセルします。

AWS マネジメントコンソール

以下で説明する手順に従って、コンソールでサービスパイプラインを更新します。

1. AWS Proton コンソールで、[Services (サービス)] を選択します。
2. サービスの一覧で、更新したいパイプラインを含むサービスの左側にあるラジオボタンを選択します。
3. サービスの詳細ページには、[Overview (概要)] と [Pipeline (パイプライン)] という 2 つのタブがあります。[Pipeline (パイプライン)] を選択します。
4. 仕様を更新したい場合、[Edit Pipeline (パイプラインの編集)] を選択して各フォームに必要な値を入力し、最終フォームが完了するまで [Next (次へ)] を選択してから [Update Pipeline (パイプラインの更新)] を選択します。

   新しいバージョンにアップデートしたい場合にパイプラインテンプレートで新しいバージョンが利用可能であることを示す情報アイコンが表示されていたら、新しいテンプレートバージョンの名前を選択します。

   a. [Update to recommended version (推奨バージョンに更新)] を選択します。
   b. 各フォームに必要な値を入力し、最終フォームが完了するまで [Next (次へ)] を選択し、[Update (更新)] を選択します。

AWS CLI

次の CLI のコマンドとレスポンスの例に示すように、サービスパイプラインを更新して新しいマイナーバージョンにします。

修正した spec でサービスパイプラインを更新する際には、"${Proton::CURRENT_VAL}" を使用して spec に値が存在する場合に元の spec を保存するパラメータ値を指定できます。get-

service で説明するように、spec を使用してサービスパイプラインの元の サービスデータを表示する を表示します。

以下の例は、"${Proton::CURRENT_VAL}" での spec の使用方法を示します。

仕様:

```
proton: ServiceSpec

pipeline:
  my_sample_pipeline_optional_input: "${Proton::CURRENT_VAL}"
  my_sample_pipeline_required_input: "${Proton::CURRENT_VAL}"

instances:
  - name: "my-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_optional_input: "${Proton::CURRENT_VAL}"
      my_sample_service_instance_required_input: "${Proton::CURRENT_VAL}"
  - name: "my-other-instance"
    environment: "simple-env"
    spec:
      my_sample_service_instance_required_input: "789"
```

コマンド: 更新する

```
$ aws proton update-service-pipeline \
    --service-name "simple-svc" \
    --spec "file://service-spec.yaml" \
    --template-major-version "1" \
    --template-minor-version "1" \
    --deployment-type "MINOR_VERSION"
```

レスポンス:

```
{
    "pipeline": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/pipeline/
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "createdAt": "2021-04-02T21:29:59.962000+00:00",
        "deploymentStatus": "IN_PROGRESS",
        "lastDeploymentAttemptedAt": "2021-04-02T21:39:28.991000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T21:29:59.962000+00:00",
```

```
        "spec": "proton: ServiceSpec\n\npipeline:\n
 my_sample_pipeline_optional_input: \"abc\"\n  my_sample_pipeline_required_input:
 \"123\"\n\ninstances:\n  - name: \"my-instance\"\n    environment: \"MySimpleEnv
\"\n    spec:\n      my_sample_service_instance_optional_input: \"def
\"\n      my_sample_service_instance_required_input: \"456\"\n  - name:
 \"my-other-instance\"\n    environment: \"MySimpleEnv\"\n    spec:\n
 my_sample_service_instance_required_input: \"789\"\n",
        "templateMajorVersion": "1",
        "templateMinorVersion": "0",
        "templateName": "svc-simple"
    }
}
```

コマンド: ステータスを取得して確認する

```
$ aws proton get-service \
    --name "simple-svc"
```

レスポンス:

```
{
    "service": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc",
        "branchName": "main",
        "createdAt": "2021-04-02T21:29:59.962000+00:00",
        "lastModifiedAt": "2021-04-02T21:30:54.364000+00:00",
        "name": "simple-svc",
        "pipeline": {
            "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/
pipeline",
            "createdAt": "2021-04-02T21:29:59.962000+00:00",
            "deploymentStatus": "SUCCEEDED",
            "lastDeploymentAttemptedAt": "2021-04-02T21:39:28.991000+00:00",
            "lastDeploymentSucceededAt": "2021-04-02T21:39:28.991000+00:00",
            "spec": "proton: ServiceSpec\n\npipeline:\n
 my_sample_pipeline_optional_input: \"abc\"\n  my_sample_pipeline_required_input:
 \"123\"\n\ninstances:\n  - name: \"instance-one\"\n    environment: \"simple-
env\"\n    spec:\n      my_sample_service_instance_optional_input: \"def
\"\n      my_sample_service_instance_required_input: \"456\"\n  - name:
 \"my-other-instance\"\n    environment: \"simple-env\"\n    spec:\n
 my_sample_service_instance_required_input: \"789\"\n",
            "templateMajorVersion": "1",
            "templateMinorVersion": "1",
```

```
            "templateName": "svc-simple"
        },
        "repositoryConnectionArn": "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "repositoryId": "repo-name/myorg-myapp",
        "spec": "proton: ServiceSpec\n\npipeline:\n
 my_sample_pipeline_optional_input: \"abc\"\n  my_sample_pipeline_required_input:
 \"123\"\n\ninstances:\n  - name: \"instance-one\"\n    environment: \"simple-
env\"\n    spec:\n      my_sample_service_instance_optional_input: \"def
\"\n      my_sample_service_instance_required_input: \"456\"\n  - name:
 \"my-other-instance\"\n    environment: \"simple-env\"\n    spec:\n
 my_sample_service_instance_required_input: \"789\"\n",
        "status": "ACTIVE",
        "templateName": "svc-simple"
    }
}
```

AWS マネジメントコンソール

次の手順に示すように、コンソールでサービスパイプラインのデプロイをキャンセルします。

1. AWS Proton コンソールのサービスナビゲーションペインで [Services (サービス)] を選択します。
2. サービスの一覧で、デプロイ更新をキャンセルしたいサービスの名前を選択します。
3. 詳細ページで [Pipeline (パイプライン)] タブを選択します。
4. 更新のデプロイステータスが [In progress (進行中)] の場合、サービスインスタンスの詳細ページで [Cancel deployment (デプロイをキャンセル)] を選択します。
5. キャンセルを確認するメッセージが表示されます。[Cancel deployment (デプロイをキャンセル)] を選択します。
6. 更新のデプロイステータスが [Cancelling (キャンセル中)] に変わり、キャンセルが完了すると [Cancelled (キャンセル済み)] に変わります。

AWS CLI

CLI のコマンドとレスポンスの例に示すように、マイナーバージョン 2 への IN_PROGRESS サービスパイプライン更新をキャンセルします。

この例で使用するテンプレートには待機条件が含まれており、更新のデプロイが成功する前にキャンセルが開始されます。

コマンド: キャンセルする

```
$ aws proton cancel-service-pipeline-deployment \
    --service-name "simple-svc"
```

レスポンス:

```
{
    "pipeline": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/pipeline",
        "createdAt": "2021-04-02T21:29:59.962000+00:00",
        "deploymentStatus": "CANCELLING",
        "lastDeploymentAttemptedAt": "2021-04-02T22:02:45.095000+00:00",
        "lastDeploymentSucceededAt": "2021-04-02T21:39:28.991000+00:00",
        "templateMajorVersion": "1",
        "templateMinorVersion": "1",
        "templateName": "svc-simple"
    }
}
```

コマンド: ステータスを取得して確認する

```
$ aws proton get-service \
    --name "simple-svc"
```

レスポンス:

```
{
    "service": {
        "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc",
        "branchName": "main",
        "createdAt": "2021-04-02T21:29:59.962000+00:00",
        "lastModifiedAt": "2021-04-02T21:30:54.364000+00:00",
        "name": "simple-svc",
        "pipeline": {
            "arn": "arn:aws:proton:region-id:123456789012:service/simple-svc/
pipeline",
            "createdAt": "2021-04-02T21:29:59.962000+00:00",
            "deploymentStatus": "CANCELLED",
            "deploymentStatusMessage": "User initiated cancellation.",
            "lastDeploymentAttemptedAt": "2021-04-02T22:02:45.095000+00:00",
            "lastDeploymentSucceededAt": "2021-04-02T21:39:28.991000+00:00",
```

```
            "spec": "proton: ServiceSpec\n\npipeline:\n
 my_sample_pipeline_optional_input: \"abc\"\n  my_sample_pipeline_required_input:
 \"123\"\n\ninstances:\n  - name: \"instance-one\"\n    environment: \"simple-
env\"\n    spec:\n      my_sample_service_instance_optional_input: \"def
\"\n      my_sample_service_instance_required_input: \"456\"\n  - name:
 \"my-other-instance\"\n    environment: \"simple-env\"\n    spec:\n
 my_sample_service_instance_required_input: \"789\"\n",
            "templateMajorVersion": "1",
            "templateMinorVersion": "1",
            "templateName": "svc-simple"
        },
        "repositoryConnectionArn": "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "repositoryId": "repo-name/myorg-myapp",
        "spec": "proton: ServiceSpec\n\npipeline:\n
 my_sample_pipeline_optional_input: \"abc\"\n  my_sample_pipeline_required_input:
 \"123\"\n\ninstances:\n  - name: \"instance-one\"\n    environment: \"simple-
env\"\n    spec:\n      my_sample_service_instance_optional_input: \"def
\"\n      my_sample_service_instance_required_input: \"456\"\n  - name:
 \"my-other-instance\"\n    environment: \"simple-env\"\n    spec:\n
 my_sample_service_instance_required_input: \"789\"\n",
        "status": "ACTIVE",
        "templateName": "svc-simple"
    }
}
```

AWS Proton コンポーネント

コンポーネントは AWS Proton リソースの一種です。コンポーネントはサービステンプレートに柔軟性を加えます。コンポーネントによって、プラットフォームチームは、コアインフラストラクチャパターンを拡張し、開発者がアプリケーションインフラストラクチャのさまざまな側面を管理するための保護手段を定義するメカニズムを利用できます。

の AWS Proton 管理者は、開発チームやアプリケーションで使用される標準インフラストラクチャを定義します。ただし、特定のユースケースに応じて、開発チームには、Amazon Simple Queue Service (Amazon SQS) キューや Amazon DynamoDB テーブルなど、追加リソースが必要になる場合があります。これらのアプリケーション固有のリソースは、特にアプリケーション開発の初期段階で頻繁に変更される可能性があります。管理者がオーサリングしたテンプレートで、このような頻繁な変化に対応するのは、管理や拡張の面で困難な場合があります。その場合、現実的な付加価値という成果は得られず、管理者には、さらに多くのテンプレートの管理が求められます。もう 1 つの方法として、アプリケーションデベロッパーがアプリケーション用にテンプレートを作成することは、AWS Fargate タスクなどの主要なアーキテクチャコンポーネントを標準化する管理者の能力を奪うため、理想的ではありません。そこで登場するのがコンポーネントです。

コンポーネントを利用すると、開発者は管理者が環境やサービステンプレートに定義したリソースをはるかに越えて、自分達が開発するアプリケーションを補うリソースを追加できます。次に、開発者はコンポーネントをサービスインスタンスにアタッチします。 は、環境とサービスインスタンスのリソースを AWS Proton プロビジョニングするのと同様に、コンポーネントによって定義されたインフラストラクチャリソースをプロビジョニングします。

コンポーネントはサービスインスタンスの入力を読み取り、サービスインスタンスに出力を提供し、完全に統合されたエクスペリエンスが実現されます。例えば、サービスインスタンスが使用するための Amazon Simple Storage Service (Amazon S3) バケットがコンポーネントによって追加されると、バケット命名の際に、コンポーネントテンプレートは環境名とサービスインスタンス名を考慮することができます。がサービステンプレートを AWS Proton レンダリングしてサービスインスタンスをプロビジョニングすると、サービスインスタンスはバケットを参照して使用できます。

AWS Proton 現在 がサポートしているコンポーネントは、直接定義されたコンポーネントです。コンポーネントのインフラストラクチャを定義する Infrastructure as Code (IaC) ファイルを AWS Proton API またはコンソールに直接渡します。これは、テンプレートバンドルで IaC を定義し、そのバンドルをテンプレートリソースとして登録し、テンプレートリソースを使用して環境またはサービスを作成する環境やサービスの場合とは異なります。

> **Note**
>
> 直接定義されたコンポーネントにより、開発者は追加のインフラストラクチャを定義し、それをプロビジョニングできます。 は、同じ AWS Identity and Access Management (IAM) ロールを使用して、同じ環境で実行されている直接定義されたすべてのコンポーネントを AWS Proton プロビジョニングします。

管理者は、開発者がコンポーネントでできる作業を次の 2 通りの方法で制御できます。

- サポートされているコンポーネントソース – 管理者は、サービステンプレートバージョンのプロパティに基づいて、 AWS Proton サービスインスタンスへのコンポーネントのアタッチを許可できます。デフォルトでは、開発者はサービスインスタンスにコンポーネントをアタッチできません。

  このプロパティの詳細については、『AWS Proton API リファレンス』の CreateServiceTemplateVersion API アクションの supportedComponentSources パラメータを参照してください。

  > **Note**
  >
  > テンプレート同期を使用する場合、リポジトリ内のサービステンプレートバンドルに変更をコミットすると、 はサービステンプレートバージョンを暗黙的に AWS Proton 作成します。その場合、サポートされているコンポーネントソースをサービステンプレートバージョンの作成時に指定する代わりに、各サービステンプレートメジャーバージョンに関連付けられたファイルにこのプロパティを指定してください。詳細については、「the section called “サービステンプレートを同期する”」を参照してください。

- コンポーネントロール – 管理者は、コンポーネントロールを環境に割り当てることができます。は、環境内で直接定義されたコンポーネントによって定義されたインフラストラクチャをプロビジョニングするときに、このロール AWS Proton を引き受けます。したがって、環境内で直接定義のコンポーネントを開発者が使用して追加できるインフラストラクチャの範囲は、このコンポーネントロールによって絞り込まれます。コンポーネントロールがないと、開発者は直接定義のコンポーネントを環境内に作成できません。

  コンポーネントロールの割り当ての詳細については、『AWS Proton API リファレンス』の CreateEnvironment API アクションの componentRoleArn パラメータを参照してください。

> **Note**
>
> コンポーネントロールは セルフマネージド型のプロビジョニング 環境では使用されません。

トピック

コンポーネントは他の AWS Proton リソースとどのように比較されますか?

多くの点で、コンポーネントは他の AWS Proton リソースと似ています。インフラストラクチャは、YAML または CloudFormation Terraform HCL 形式で作成された IaC テンプレートファイルで定義されます。 は、 AWSマネージドプロビジョニングまたはセルフマネージドプロビジョニングを使用してコンポーネントインフラストラクチャをプロビジョニング AWS Proton できます。

ただし、コンポーネントは、いくつかの点で他の AWS Proton リソースとは異なります。

- デタッチ状態 — コンポーネントは、サービスインスタンスにアタッチしてそのインフラストラクチャを拡張するように設計されていますが、どのサービスインスタンスにもアタッチされていないデタッチ状態も可能です。コンポーネントのステータスの詳細については、「the section called “コンポーネントの状態”」を参照してください。
- スキーマなし — コンポーネントには、テンプレートバンドルにあるような関連スキーマはありません。コンポーネント入力はサービスによって定義されます。コンポーネントは、サービスインスタンスにアタッチされると入力を消費できます。
- カスタマー管理のコンポーネントはありません。 AWS Proton 常にコンポーネントインフラストラクチャをプロビジョニングします。コンポーネントにはあなた自身のリソースを持ち込むバー

ジョンはありません。お客様による管理環境の詳細については、「the section called “作成”」を参照してください。

- テンプレートリソースなし — 直接定義のコンポーネントには、環境やサービステンプレートと同様、関連テンプレートリソースはありません。IaC テンプレートファイルは、あなたがコンポーネントに直接提供します。同様に、コンポーネントのインフラストラクチャをプロビジョニングするためのテンプレート言語とレンダリングエンジンを定義するマニフェストはあなたが直接提供します。テンプレートファイルとマニフェストは、テンプレートバンドルのオーサリングと同様の方法であなたがオーサリングします。ただし、コンポーネントが直接定義されている場合、IaC ファイルを特定の場所にバンドルとして保存する必要はありません。また、IaC ファイル AWS Proton から にテンプレートリソースを作成する必要はありません。
- CodeBuild ベースのプロビジョニングなし — CodeBuild ベースのプロビジョニングと呼ばれるあなた自身のカスタムプロビジョニングスクリプトで、あなたが直接定義のコンポーネントをプロビジョニングすることはできません。詳細については、「the section called “CodeBuild プロビジョニング”」を参照してください。

AWS Proton コンソールのコンポーネント

AWS Proton コンソールを使用して、 AWS Proton コンポーネントを作成、更新、表示、および使用します。

次のコンソールページはコンポーネントに関連があります。ここには、トップレベルのコンソールページまでの直接リンクがあります。

- コンポーネント – AWS アカウントのコンポーネントのリストを表示します。新しいコンポーネントの作成や、既存のコンポーネントの更新、または削除ができます。コンポーネントの詳細ページを表示するには、コンポーネント名を選択します。

  [環境の詳細] ページと [サービスインスタンスの詳細] ページにも同様のリストがあります。これらのリストには、現在表示されているリソースに関連するコンポーネントのみが表示されます。これらのリストからコンポーネントを作成すると、 はコンポーネントの作成ページで関連する環境を AWS Proton 事前に選択します。
- コンポーネントの詳細 — コンポーネントの詳細ページを表示するには、[コンポーネント] リストでコンポーネント名を選択します。

  詳細ページで、コンポーネントの詳細とステータスを表示し、コンポーネントを更新するか、削除します。出力 (プロビジョニングされたリソース ARNs など) 、プロビジョニングされた CloudFormation スタック、割り当てられたタグのリストを表示および管理します。

- コンポーネントを作成 — コンポーネントを作成します。コンポーネント名と説明を入力し、関連するリソースを選択し、コンポーネントソース IaC ファイルを指定して、タグを割り当てます。
- コンポーネントの更新 — コンポーネントを更新するには、[コンポーネント] リストでそのコンポーネントを選択し、[アクション] メニューで [コンポーネントの更新] を選択します。または、[コンポーネント詳細] ページで [更新] を選択します。

  あなたはリージョンを更新できません。あなたはリージョンは更新できません。また、更新が成功した後にコンポーネントを再デプロイするかどうかはあなたが選択できます。
- 環境設定 — 環境の作成時や更新時には、コンポーネントロールを指定できます。このロールは、環境内で直接定義のコンポーネントを実行する機能を制御します。またそれらの機能をプロビジョニングするための権限がこのロールで与えられます。
- 新しいサービステンプレートバージョンの作成 — サービステンプレートバージョンの作成時に、そのテンプレートバージョンの「サポート対象のコンポーネントソース」を指定できます。これにより、サービスのサービスインスタンスにコンポーネントをアタッチする機能が、このテンプレートバージョンに基づいて制御されます。

AWS Proton API および のコンポーネント AWS CLI

AWS Proton API または AWS CLI を使用して、 AWS Proton コンポーネントを作成、更新、表示、使用します。

次の API アクションは、 AWS Proton コンポーネントリソースを直接管理します。

- CreateComponent — AWS Proton コンポーネントを作成します。
- DeleteComponent — AWS Proton コンポーネントを削除します。
- GetComponent — コンポーネントの詳細データを取得します。
- ListComponentOutputs — コンポーネントの一覧 Infrastructure as Code (IaC) 出力を取得します。
- ListComponentProvisionedResources — コンポーネントにプロビジョニングされたリソースを詳細とともに一覧表示します。
- ListComponents — コンポーネントをサマリーデータと共に一覧表示します。結果リストは、環境やサービスごとに、または 1 つのサービスインスタンスでフィルタリングできます。

他の AWS Proton リソースの次の API アクションには、コンポーネントに関連するいくつかの機能があります。

- CreateEnvironment、UpdateEnvironment – この環境で直接定義されたコンポーネントをプロビジョニングするときに が AWS Proton 使用する IAM サービスロールの Amazon リソースネーム (ARN) componentRoleArnを指定します。直接定義のコンポーネントでプロビジョニングできるインフラストラクチャの範囲がこれで決定します。
- CreateServiceTemplateVersion — サポートされるコンポーネントソースは、supportedComponentSources で指定します。サポートされるソースがあるコンポーネントは、このサービステンプレートバージョンに基づいてサービスインスタンスにアタッチできます。

コンポーネントに関するよくある質問

コンポーネントのライフサイクルはどのようになっていますか?

コンポーネントはアタッチされた状態でもデタッチされた状態でもかまいません。コンポーネントは、ほとんどの時間、サービスインスタンスにアタッチしてインフラストラクチャを強化する設計になっています。切り離されたコンポーネントは移行状態なので、制御された安全な方法で、コンポーネントの削除や、別のサービスインスタンスへのアタッチができます。詳細については、「the section called “コンポーネントの状態”」を参照してください。

アタッチしたコンポーネントを削除できないのはなぜですか?

解決策:アタッチされているコンポーネントを削除するには、コンポーネントを更新してサービスインスタンスから切り離し、サービスインスタンスの安定性を確認してから、コンポーネントを削除します。

なぜこれが必要なのですか? アタッチされたコンポーネントは、あなたのアプリケーションがランタイム機能を実行するのに必要な追加のインフラストラクチャを提供します。サービスインスタンスがコンポーネント出力を使用して、このインフラストラクチャのリソースを検出して使用している可能性があります。コンポーネントを削除して、そのインフラストラクチャリソースを削除すると、アタッチされているサービスインスタンスに影響が及ぶおそれがあります。

追加の安全対策として、 AWS Proton では、コンポーネントを削除する前に、コンポーネントを更新してサービスインスタンスからデタッチする必要があります。その後、あなたのサービスインスタンスを検証すれば、引き続き適切にデプロイされて機能することを確認できます。問題が見つかった場合は、すぐにコンポーネントをサービスインスタンスに再アタッチすれば、問題の修正に取り組むことができます。あなたのサービスインスタンスがコンポーネントに依存していないことが確認できたら、コンポーネントを安全に削除できます。

コンポーネントにアタッチされているサービスインスタンスを直接変更できないのはなぜですか?

解決策:アタッチメントを変更するには、コンポーネントを更新してサービスインスタンスからデタッチし、コンポーネントとサービスインスタンスの安定性を確認してから、コンポーネントを新しいサービスインスタンスにアタッチします。

なぜこれが必要なのですか? コンポーネントはサービスインスタンスにアタッチされる設計になっています。インフラストラクチャリソースの命名と設定にサービスインスタンス入力が、あなたのコンポーネントによって使用される場合があります。アタッチされているサービスインスタンスを変更すると、コンポーネントが破壊されるおそれがあります (前の FAQ「アタッチしたコンポーネントを削除できないのはなぜですか?」で説明したように、サービスインスタンスの破壊と併せて)。アタッチされた状態では、たとえば、コンポーネントの IaC テンプレートで定義されているリソースの名前が変更され、場合によっては置き換えられるおそれがあります。

追加の安全対策として、 では、別のサービスインスタンスにアタッチする前に、コンポーネントを更新し、そのサービスインスタンスからデタッチ AWS Proton する必要があります。そのため、コンポーネントを新しいサービスインスタンスにアタッチする前に、コンポーネントとサービスインスタンスの両方の安定性を検証できます。

コンポーネントの状態

AWS Proton コンポーネントは、基本的に異なる 2 つの状態になります。

- アタッチ済み — コンポーネントはサービスインスタンスにアタッチされています。コンポーネントは、サービスインスタンスのランタイム機能をサポートするインフラストラクチャを定義します。コンポーネントは、環境とサービステンプレートで定義されたインフラストラクチャを開発者が定義したインフラストラクチャで拡張します。

  一般的なコンポーネントは、ライフサイクルで利用できる期間のほとんどでアタッチされた状態になっています。
- Detached – コンポーネントは AWS Proton 環境に関連付けられており、環境内のサービスインスタンスにはアタッチされません。

  これは、コンポーネントの有効期間を 1 つのサービスインスタンス以上に延長するための移行状態です。

次の表は、さまざまなコンポーネントの状態をトップレベルで比較したものです。

	アタッチ済み	デタッチ済み
状態の主な目的	サービスインスタンスのインフラストラクチャを拡張する。	サービスインスタンスアタッチメント間のコンポーネントのインフラストラクチャを維持する。
関連付け	サービスインスタンスと環境	環境
主な特有のプロパティ	• サービス名 • サービスインスタンス名 • 仕様	• 環境名
削除可能	✗いいえ	✓ はい
別のサービスインスタンスに更新可能	✗いいえ	✓ はい
入力の読み取り可能	✓ はい	✗いいえ

コンポーネントの主な目的は、サービスインスタンスにアタッチして、追加リソースでそのインフラストラクチャを拡張することです。アタッチされたコンポーネントは、仕様に従ってサービスインスタンスから入力を読み取ることができます。コンポーネントは、直接削除したり、別のサービスインスタンスにアタッチしたりすることはできません。そのサービスインスタンス、または関連するサービスと環境も削除できません。これらの操作を行うには、まず、コンポーネントを更新して、そのサービスインスタンスからデタッチします。

1 つのサービスインスタンスの存続期間を超えてコンポーネントのインフラストラクチャを維持するには、コンポーネントを更新し、サービスとサービスインスタンス名を削除してサービスインスタンスからデタッチします。このデタッチ状態は移行状態です。このコンポーネントには入力がありません。そのインフラストラクチャはプロビジョニングされたままであり、更新できます。コンポーネントがアタッチされたときに関連付けられていたリソース (サービスインスタンス、サービス) は削除できます。コンポーネントは削除することも、更新してサービスインスタンスに再度アタッチすることもできます。

Component Infrastructure as Code ファイル

コンポーネントInfrastructure as Code (IaC) ファイルは、他の AWS Proton リソースのファイルと似ています。コンポーネント固有の詳細についてはこちらをご覧ください。の IaC ファイルの作成の詳細については AWS Proton、「」を参照してくださいテンプレートのオーサリングとバンドル。

コンポーネントでのパラメータの使用

AWS Proton パラメータ名前空間には、関連するコンポーネントの名前と出力を取得するためにサービス IaC ファイルが参照できるいくつかのパラメータが含まれています。名前空間には、コンポーネントが関連付けられている環境、サービス、およびサービスインスタンスから入力、出力、およびリソース値を取得するためにコンポーネント IaC ファイルが参照できるパラメーターもあります。

コンポーネントに専用の入力はありません。アタッチ先のサービスインスタンスから入力を取得します。コンポーネントは環境出力も読み取ることができます。

コンポーネントと関連サービスの IaC ファイルにおけるパラメータの使用方法については、the section called “コンポーネント CloudFormation IaC パラメータ” を参照してください。 AWS Proton パラメータの一般的な情報とパラメータ名前空間の完全なリファレンスについては、「」を参照してくださいthe section called “パラメータ”。

堅牢な IaC ファイルのオーサリング

あなたは、サービステンプレートバージョンの作成時に、そのテンプレートバージョンから作成したサービスインスタンスへのコンポーネントのアタッチを許可するかどうかを決定できます。AWS Proton API リファレンスの CreateServiceTemplateVersion API アクションの supportedComponentSources パラメータを参照してください。ただし、将来のサービスインスタンスでは、インスタンスを作成し、コンポーネントをそのインスタンスにアタッチするかどうかを決定し、(直接定義のコンポーネントの場合) コンポーネント IaC をオーサリングするユーザーは、通常、別の人、すなわち、あなたのサービステンプレートを使用する開発者です。したがって、コンポーネントがサービスインスタンスにアタッチされることを開発者は保証できません。また、特定のコンポーネント出力名の存在や、これらの出力の値の有効性と安全性を保証することもできません。

AWS Proton と Jinja 構文は、これらの問題を回避し、次の方法で障害なくレンダリングされる堅牢なサービステンプレートを作成するのに役立ちます。

- AWS Proton パラメータフィルター – コンポーネント出力プロパティを参照する場合、パラメータフィルターを使用できます。パラメータフィルターは、パラメータ値を検証、フィルタリング、

フォーマットする修飾子です。詳細な説明と例については、the section called “CloudFormation パラメータフィルター” を参照してください。

- 単一プロパティデフォルト — コンポーネントの 1 つのリソースや出力プロパティを参照する場合、デフォルト値の有無にかかわらず、`default` フィルターを使用すればあなたのサービステンプレートによるレンダリングが失敗することはありません。コンポーネント、または参照している特定の出力パラメーターが存在しない場合、代わりにデフォルト値 (デフォルト値を指定していない場合は空の文字列) がレンダリングされ、レンダリングは成功します。詳細については、「the section called “デフォルト値を指定します。”」を参照してください。

  例:

  - `{{ service_instance.components.default.name | default("") }}`
  - `{{ service_instance.components.default.outputs.my-output | default("17") }}`

  > **Note**
  >
  > 直接定義のコンポーネントを指定する名前空間の `.default` 部分を、参照するプロパティが存在しない場合にデフォルト値を提供する `default` フィルターと混同しないでください。

- オブジェクト参照全体 — コンポーネント全体、またはコンポーネントの出力のコレクションを参照すると、 AWS Proton で空のオブジェクト `{}` が戻るので、あなたのサービステンプレートのレンダリングが失敗することはありません。追加のアクションを実行する必要はありません。参照先は必ず空のオブジェクトを取得できるコンテキストで行うか、または `{{ if .. }}` 条件で、空のオブジェクトをテストしてください。

  例:

  - `{{ service_instance.components.default }}`
  - `{{ service_instance.components.default.outputs }}`

コンポーネントの CloudFormation 例

AWS Proton 直接定義されたコンポーネントの完全な例と、それを AWS Proton サービスで使用する方法を示します。このコンポーネントによって Amazon Simple Storage Service (Amazon S3) バケットと関連アクセス ポリシーのプロビジョンが行われます。サービスインスタンスはこのバケットを参照して使用できます。バケット名は環境、サービス、サービスインスタンス、およびコンポー

ネントの名前に基づいています。言い換えると、バケットはコンポーネントテンプレートの特定のインスタンスと結合して、特定のサービスインスタンスを拡張します。開発者は、このコンポーネントテンプレートに基づいて複数のコンポーネントを作成し、さまざまなサービスインスタンスや機能上のニーズに合わせて Amazon S3 バケットをプロビジョニングできます。

この例では、さまざまな必要な CloudFormation Infrastructure as Code (IaC) ファイルの作成と、必要な AWS Identity and Access Management (IAM) ロールの作成について説明します。この例では、オーナーロールごとにステップをグループ分けしています。

管理者のステップ

開発者がサービスでコンポーネントを使用できるようにする

1. 環境で実行されているコンポーネントを直接定義したリソースをスコープダウンする AWS Identity and Access Management (IAM) ロールを作成します。 は後でこのロールを AWS Proton 引き受けて、環境で直接定義されたコンポーネントをプロビジョニングします。

   この例では、次のポリシーを使用する必要があります。

   Example直接定義されるコンポーネントロール

   JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudformation:CancelUpdateStack",
        "cloudformation:CreateChangeSet",
        "cloudformation:DeleteChangeSet",
        "cloudformation:DescribeStacks",
        "cloudformation:ContinueUpdateRollback",
        "cloudformation:DetectStackResourceDrift",
        "cloudformation:DescribeStackResourceDrifts",
        "cloudformation:DescribeStackEvents",
        "cloudformation:CreateStack",
        "cloudformation:DeleteStack",
        "cloudformation:UpdateStack",
        "cloudformation:DescribeChangeSet",
```

```
        "cloudformation:ExecuteChangeSet",
        "cloudformation:ListChangeSets",
        "cloudformation:ListStackResources"
      ],
      "Resource": "arn:aws:cloudformation:*:123456789012:stack/AWSProton-*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:CreateBucket",
        "s3:DeleteBucket",
        "s3:GetBucket*",
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:GetPolicy",
        "iam:ListPolicyVersions",
        "iam:DeletePolicyVersion"
      ],
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws:CalledVia": "cloudformation.amazonaws.com"
        }
      }
    }
  ]
}
```

2. 前のステップで作成した設定をデプロイするときに前のステップで作成した設定を渡して、前のステップで作成した設定を渡して、前のステップで作成した設定をデプロイします。 AWS Proton コンソールで、環境の設定ページでコンポーネントロールを指定します。 AWS Proton API または を使用している場合は AWS CLI、CreateEnvironment または UpdateEnvironment API アクションcomponentRoleArnの を指定します。

3. サービスインスタンスにアタッチされた、直接定義のコンポーネントを参照するサービステンプレートを作成します。

   この例では、コンポーネントがサービスインスタンスにアタッチされていなくても壊れない堅牢なサービステンプレートを作成する方法を示しています。

   Exampleコンポーネントで CloudFormation IaC ファイルをサービスする

   ```
   # service/instance_infrastructure/cloudformation.yaml
   ```

```
Resources:
  TaskDefinition:
    Type: AWS::ECS::TaskDefinition
    Properties:
      TaskRoleArn: !Ref TaskRole
      ContainerDefinitions:
        - Name: '{{service_instance.name}}'
          # ...
          {% if service_instance.components.default.outputs | length > 0 %}
          Environment:
            {{ service_instance.components.default.outputs |
                proton_cfn_ecs_task_definition_formatted_env_vars }}
          {% endif %}

  # ...

  TaskRole:
    Type: AWS::IAM::Role
    Properties:
      # ...
      ManagedPolicyArns:
        - !Ref BaseTaskRoleManagedPolicy
        {{ service_instance.components.default.outputs
            | proton_cfn_iam_policy_arns }}

  # Basic permissions for the task
  BaseTaskRoleManagedPolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      # ...
```

4. 直接定義のコンポーネントをサポート対象と宣言する新しいサービステンプレートマイナーバージョンを作成します。

   - Amazon S3 のテンプレートバンドル – AWS Proton コンソールで、サービステンプレートバージョンを作成するときに、サポートされているコンポーネントソースで、直接定義を選択します。 AWS Proton API または を使用している場合は AWS CLI、CreateServiceTemplateVersion または UpdateServiceTemplateVersion API アクションの `supportedComponentSources`パラメータ`DIRECTLY_DEFINED`で を指定します。
   - テンプレート同期 — メジャーバージョンディレクトリの `.template-registration.yaml` ファイル内の `supported_component_sources:` 項目として

DIRECTLY_DEFINED を指定して、あなたのサービステンプレートバンドルリポジトリに変更をコミットします。ファイルの詳細については、「the section called “サービステンプレートを同期する”」を参照してください。

5. 新規サービステンプレートのマイナーバージョンをパブリッシュする。詳細については、「the section called “配信”」を参照してください。
6. このサービステンプレートを使用する開発者の IAM ロールで、必ず proton:CreateComponent を許可してください。

開発者向けステップ

直接定義のコンポーネントをサービスインスタンスで使用するには

1. コンポーネントサポートで管理者が作成したサービステンプレートバージョンを使用するサービスを作成します。または、あなたの既存のサービスインスタンスの 1 つを更新して、最新のテンプレートバージョンを使用します。
2. Amazon S3 バケットと関連するアクセスポリシーをプロビジョニングし、これらのリソースを出力として公開するコンポーネント IaC テンプレートファイルを作成します。

   Exampleコンポーネント CloudFormation IaC ファイル

```
# cloudformation.yaml

# A component that defines an S3 bucket and a policy for accessing the bucket.
Resources:
  S3Bucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: '{{environment.name}}-{{service.name}}-{{service_instance.name}}-
{{component.name}}'
  S3BucketAccessPolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      PolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Action:
              - 's3:Get*'
              - 's3:List*'
              - 's3:PutObject'
```

```
            Resource: !GetAtt S3Bucket.Arn
Outputs:
  BucketName:
    Description: "Bucket to access"
    Value: !GetAtt S3Bucket.Arn
  BucketAccessPolicyArn:
    Value: !Ref S3BucketAccessPolicy
```

3. AWS Proton API または を使用している場合は AWS CLI、コンポーネントのマニフェストファイルを書き込みます。

   Example直接定義のコンポーネントマニフェスト

   ```
   infrastructure:
     templates:
       - file: "cloudformation.yaml"
         rendering_engine: jinja
         template_language: cloudformation
   ```

4. 直接定義されたコンポーネントを作成します。 は、管理者がコンポーネントをプロビジョニングするために定義したコンポーネントロール AWS Proton を引き受けます。

   AWS Proton コンソールのコンポーネントページで、コンポーネントの作成を選択します。[コンポーネント設定] には、[コンポーネント名] と、オプションで [コンポーネントの説明] を入力します。[コンポーネントのアタッチメント] で、[コンポーネントをサービスインスタンスにアタッチ] を選択します。あなたの環境、サービス、サービスインスタンスを選択します。[コンポーネントソース] で、[CloudFormation] を選択し、次にコンポーネント IaC ファイルを選択します。

   > **Note**
   >
   > マニフェストを作成する必要はありません。コンソールが作成します。

   AWS Proton API または を使用している場合は AWS CLI、CreateComponent API アクションを使用します。`name` コンポーネントとオプションの `description` を設定します。`environmentName`、`serviceName`、`serviceInstanceName` を設定します。作成したファイルまでのパスに `templateSource` と `manifest` を設定します。

> **Note**
>
> サービスとサービスインスタンス名を指定するとき、環境名の指定は任意です。これら 2 つの組み合わせは AWS アカウント内で一意であり、サービスインスタンスから環境を判断 AWS Proton できます。

5. サービスインスタンスを更新して再デプロイします。 は、レンダリングされたサービスインスタンステンプレートのコンポーネントからの出力 AWS Proton を使用して、コンポーネントがプロビジョニングした Amazon S3 バケットをアプリケーションが使用できるようにします。

での git リポジトリの使用 AWS Proton

AWS Proton は、さまざまな目的で git リポジトリを使用します。次のリストは、 AWS Proton リソースに関連付けられているリポジトリタイプを分類します。リポジトリに繰り返し接続してコンテンツをプッシュするか、そこからコンテンツをプルする AWS Proton 機能については、 AWS Proton AWS アカウントにリポジトリリンクを に登録する必要があります。リポジトリリンクは、リポジトリに接続するときに AWS Proton が使用できる一連のプロパティです。 AWS Proton は現在、GitHub、GitHub Enterprise、BitBucket をサポートしています。

開発者リポジトリ

コードリポジトリ — 開発者がアプリケーションコードの保存に使用するリポジトリ。コードデプロイに使用されます。このリポジトリと直接やり取り AWS Proton しません。開発者がパイプラインのあるサービスをプロビジョニングするとき、アプリケーションコードの読み取り元になるリポジトリ名とブランチを指定します。 AWS Proton はこの情報をプロビジョニングするパイプラインに渡します。

詳細については、「the section called “作成”」を参照してください。

管理者用リポジトリ

テンプレートリポジトリ – 管理者が AWS Proton テンプレートバンドルを保存するリポジトリ。テンプレート同期に使用します。管理者が でテンプレートを作成すると AWS Proton、テンプレートリポジトリをポイントし、新しいテンプレートを同期 AWS Proton させることができます。管理者がリポジトリ内のテンプレートバンドルを更新すると、 は新しいテンプレートバージョン AWS Proton を自動的に作成します。同期に使用する AWS Proton 前に、テンプレートリポジトリを にリンクします。

詳細については、「the section called “テンプレートの同期設定”」を参照してください。

> **Note**
>
> テンプレートを Amazon Simple Storage Service (Amazon S3) にアップロードし続け、AWS Proton テンプレート管理 APIs を呼び出して新しいテンプレートまたはテンプレートバージョンを作成する場合、テンプレートリポジトリは必要ありません。

セルフマネージドプロビジョニングレポジトリ

インフラストラクチャリポジトリ — レンダリングされたインフラストラクチャテンプレートをホストするリポジトリです。リソースインフラストラクチャのセルフマネージドプロビジョニングに使用します。管理者がセルフマネージドプロビジョニング用の環境を作成すると、リポジトリが提供されます。 はこのリポジトリにプルリクエスト (PRs) AWS Proton を送信して、環境と環境にデプロイされたサービスインスタンスのインフラストラクチャを作成します。セルフマネージドインフラストラクチャのプロビジョニングに使用する前に、インフラストラクチャリポジトリを AWS Proton にリンクします。

パイプラインリポジトリ — パイプラインの作成に使用するリポジトリです。パイプラインのセルフマネージドプロビジョニングに使用します。追加のリポジトリを使用してパイプラインをプロビジョニングすると、 AWS Proton は個々の環境またはサービスと独立してパイプライン設定を保存できます。あなたのセルフマネージドプロビジョニングサービスのすべてに 1 つのパイプラインリポジトリを用意するだけで済みます。自己管理型パイプラインのプロビジョニングに使用する AWS Proton 前に、パイプラインリポジトリを にリンクします。

詳細については、「the section called “AWSマネージドプロビジョニング”」を参照してください。

トピック

あなたのリポジトリのリンクを作成して登録する

コンソールまたは CLI で、あなたのリポジトリのリンクを作成できます。リポジトリリンクを作成すると、 によってサービスにリンクされたロール AWS Proton が作成されます。

AWS マネジメントコンソール

次のコンソール手順に示すように、あなたのリポジトリのリンクを作成します。

1. AWS Proton コンソールで [Repositories (リポジトリ)] を選択します。
2. [Create repository (リポジトリの作成)] を選択します。

3. [Link new repository (新しいリポジトリをリンクする)] ページの [Repository details (リポジトリの詳細)] セクションで以下の手順に従います。

   a. あなたのリポジトリプロバイダを選択します。

   b. 既存のあなたの接続を 1 つ選択してください。接続がない場合は、新しい CodeStar 接続を追加を選択して接続を作成し、 AWS Proton コンソールに戻り、接続リストを更新して、新しい接続を選択します。

   c. 接続しているソースコードリポジトリのいずれかを選択します。

4. [オプション] [タグ]セクションで、[新しいタグを追加] を 1 回以上選択し、キーと値のペアを入力します。

5. [Create repository (リポジトリの作成)] を選択します。

6. リンクされたリポジトリの詳細データを表示します。

AWS CLI

リポジトリのリンクを作成して登録します。

次のコマンドを実行してください。

```
$ aws proton create-repository \
    --name myrepos/environments \
    --connection-arn "arn:aws:codestar-connections:region-
id:123456789012:connection/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111" \
    --provider "GITHUB" \
    --encryption-key "arn:aws:kms:region-id:123456789012:key/bPxRfiCYEXAMPLEKEY" \
    --tags key=mytag1,value=value1 key=mytag2,value=value2
```

最後の --encryption-key および --tags パラメータはどちらも任意です。

レスポンス:

```
{
    "repository": {
        "arn": "arn:aws:proton:region-id:123456789012:repository/github:myrepos/
environments",
        "connectionArn": "arn:aws:codestar-connections:region-
id:123456789012:connection/2ad03b28-a7c4-EXAMPLE11111",
        "encryptionKey": "arn:aws:kms:region-id:123456789012:key/
bPxRfiCYEXAMPLEKEY",
```

```
        "name": "myrepos/environments",
        "provider": "GITHUB"
    }
}
```

リポジトリリンクを作成したら、次の コマンド例に示すように、 AWS とカスタマーマネージドタグのリストを表示できます。 は AWS マネージドタグ AWS Proton を自動的に生成します。また、 AWS CLIで、カスタマーマネージドタグの変更や作成もできます。詳細については、「AWS Proton リソースとタグ付け」を参照してください。

コマンド:

```
$ aws proton list-tags-for-resource \
    --resource-arn "arn:aws:proton:region-id:123456789012:repository/github:myrepos/environments"
```

リンクされたリポジトリデータを表示する

コンソールまたは AWS CLIを使用して、リポジトリのリストや詳細データを表示できます。git リポジトリを と同期するために使用されるリポジトリリンクについては AWS Proton、 を使用してリポジトリの同期定義とステータスを取得できます AWS CLI。

AWS マネジメントコンソール

AWS Proton コンソールで、リンクされたリポジトリのリストや詳細を表示します。

1. あなたのリンクされたリポジトリを一覧表示するには、ナビゲーションペインで [Repositories (リポジトリ)] を選択します。
2. 詳細データを表示するには、リポジトリの名前を選択します。

AWS CLI

あなたのリンクされているリポジトリを一覧表示します。

次のコマンドを実行してください。

```
$ aws proton list-repositories
```

レスポンス:

```
{
    "repositories": [
        {
            "arn": "arn:aws:proton:region-id:123456789012:repository/github:myrepos/
templates",
            "name": "myrepos/templates",
            "provider": "GITHUB"
        },
        {
            "arn": "arn:aws:proton:region-id:123456789012:repository/github:myrepos/
environments",
            "name": "myrepos/environments",
            "provider": "GITHUB"
        }
    ]
}
```

リンクされたリポジトリの詳細情報を表示します。

次のコマンドを実行してください。

```
$ aws proton get-repository \
    --name myrepos/templates \
    --provider "GITHUB"
```

レスポンス:

```
{
    "repository": {
        "arn": "arn:aws:proton:region-id:123456789012:repository/github:myrepos/
templates",
        "name": "myrepos/templates",
        "provider": "GITHUB"
    }
}
```

あなたの同期したリポジトリが一覧表示される。

次の例では、テンプレート同期用に設定したリポジトリを一覧表示します。

次のコマンドを実行してください。

```
$ aws proton list-repository-sync-definitions \
    --branch "main" \
    --repository-name myrepos/templates \
    --repository-provider "GITHUB" \
    --sync-type "TEMPLATE_SYNC"
```

リポジトリの同期ステータスを表示します。

次の例では、テンプレート同期リポジトリの同期ステータスを取得します。

次のコマンドを実行してください。

```
$ aws proton get-repository-sync-status \
    --branch "main" \
    --repository-name myrepos/templates \
    --repository-provider "GITHUB" \
    --sync-type "TEMPLATE_SYNC"
```

レスポンス:

```
{
    "latestSync": {
        "events": [
            {
                "event": "Clone started",
                "time": "2021-11-21T00:26:35.883000+00:00",
                "type": "CLONE_STARTED"
            },
            {
                "event": "Updated configuration",
                "time": "2021-11-21T00:26:41.894000+00:00",
                "type": "CONFIG_UPDATED"
            },
            {
                "event": "Starting syncs for commit 62c03ff86eEXAMPLE1111111",
                "externalId": "62c03ff86eEXAMPLE1111111",
                "time": "2021-11-21T00:26:44.861000+00:00",
                "type": "STARTING_SYNC"
            }
        ],
        "startedAt": "2021-11-21T00:26:29.728000+00:00",
        "status": "SUCCEEDED"
```

```
    }
}
```

リポジトリリンクを削除する

コンソールまたは AWS CLIを使用して、リポジトリリンクを削除できます。

> **Note**
>
> リポジトリリンクを削除すると、アカウントに AWS Proton AWS がある登録済みリンクのみが削除されます。このとき、あなたのリポジトリからは、いかなる情報も削除されません。

AWS マネジメントコンソール

コンソールでリポジトリリンクを削除します。

リポジトリの詳細ページで以下の操作をします。

1. AWS Proton コンソールで [Repositories (リポジトリ)] を選択します。
2. スナップショットのリストで、削除したいスナップショットの左にあるボタンを選択します。
3. [Delete (削除)] を選択します。
4. モーダルから削除アクションの確認を求めるプロンプトが表示されます。
5. 指示に従って操作し、[Yes, delete (はい、削除します)] を選択します。

AWS CLI

リポジトリリンクを削除します。

次のコマンドを実行してください。

```
$ aws proton delete-repository \
    --name myrepos/templates \
    --provider"GITHUB"
```

レスポンス:

```
{
    "repository": {
        "arn": "arn:aws:proton:region-id:123456789012:repository/github:myrepos/
templates",
        "name": "myrepos/templates",
        "provider": "GITHUB"
    }
}
```

モニタリング AWS Proton

モニタリングは、 および AWS Proton AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要な部分です。次のセクションでは、 で使用できるモニタリングツールについて説明します AWS Proton。

EventBridge AWS Proton による自動化

Amazon EventBridge で AWS Proton イベントをモニタリングできます。EventBridge は、独自のアプリケーション、software-as-a-service (SaaS) アプリケーション、および からリアルタイムデータのストリームを提供します AWS のサービス。 AWS リソースの状態の変化に応答するようにイベントを設定できます。EventBridge はこのデータを AWS Lambda や Amazon Simple Notification Service などのターゲットサービスにルーティングします。これらのイベントは、Amazon CloudWatch Events に表示されるイベントと同じです。CloudWatch Events は、 AWS リソースの変更を記述するシステムイベントのほぼリアルタイムのストリームを提供します。詳細については、Amazon EventBridge ユーザーガイドの「What Is Amazon EventBridge?」を参照してください。

EventBridge を使用して、 AWS Proton プロビジョニングワークフローの状態変更を通知します。

イベントタイプ

イベントは、イベントパターンとターゲットを含むルールで構成されます。ルールを設定するには、イベントパターンとターゲットオブジェクトを選択します。

イベントパターン

各ルールは、モニタリングするイベントのソースとタイプ、およびイベントターゲットを含むイベントパターンとして表現されます。イベントをモニタリングするには、モニタリングしたいサービスをイベントソースとしてルールを作成します。たとえば、 AWS Proton をイベントソースとして使用するルールをイベントパターンとともに作成し、デプロイメント状態に変更があった場合にルールをトリガーすることが可能です。

ターゲット

ルールは選択したサービスをイベントターゲットとして受け取ります。ターゲットサービスを設定することで、通知を送り、状態情報を取得し、是正措置を講じ、イベントを開始し、その他のアクションを実行できます。

イベントオブジェクトには、ID、アカウント、詳細タイプ AWS リージョン、ソース、バージョン、リソース、時間 (オプション) の標準フィールドが含まれます。詳細フィールドは、イベントのカスタムフィールドを含むネストされたオブジェクトです。

AWS Proton イベントはベストエフォートベースで出力されます。ベストエフォート配信の場合、すべてのイベントを EventBridge に送信しようとしますが、まれにイベントが配信されない場合があります。

イベントを出力できる各 AWS Proton リソースについて、次の表に詳細タイプ値、詳細フィールド、および (利用可能な場合) `status`および`previousStatus`詳細フィールドの値のリストへの参照を示します。リソースを削除すると、`status` 詳細フィールド値が `DELETED`になります。

[リソース]	詳細タイプ値	詳細フィールド

`EnvironmentTemplate`	AWS Proton 環境テンプレートのステータスの変更	`name`
		`status`
		`previousStatus`

`EnvironmentTemplateVersion`	AWS Proton 環境テンプレートのバージョンステータスの変更	`name`
		`majorVersion`
		`minorVersion`
		`status`
		`previousStatus`
		ステータス値

`ServiceTemplate`	AWS Proton サービステンプレートのステータスの変更	`name` `status`

[リソース]	詳細タイプ値	詳細フィールド
		previousStatus
ServiceTemplateVersion	AWS Proton サービステンプレートのバージョンステータスの変更	name majorVersion minorVersion status previousStatus ステータス値
Environment	AWS Proton 環境ステータスの変更	name status previousStatus
Service	AWS Proton サービスステータスの変更	name status previousStatus ステータス値

[リソース]	詳細タイプ値	詳細フィールド
ServiceInstance	AWS Proton サービスインスタンスのステータスの変更	name serviceName status previousStatus
ServicePipeline	AWS Proton サービスパイプラインのステータスの変更	serviceName status previousStatus
EnvironmentAccountConnection	AWS Proton 環境アカウント接続ステータスの変更	id status previousStatus ステータス値
Component	AWS Proton コンポーネントステータスの変更	name status previousStatus

AWS Proton イベントの例

次の例は、 AWS Proton が EventBridge にイベントを送信する方法を示しています。

サービステンプレート

```
{
```

```
    "source": "aws.proton",
    "detail-type": ["AWS Proton Service Template Status Change"],
    "time": "2021-03-22T23:21:40.734Z",
    "resources": ["arn:aws:proton:region_id:123456789012:service-template/sample-
service-template-name"],
    "detail": {
        "name": "sample-service-template-name",
        "status": "PUBLISHED",
        "previousStatus": "DRAFT"
    }
}
```

サービステンプレートバージョン

```
{
    "source": "aws.proton",
    "detail-type": ["AWS Proton Service Template Version Status Change"],
    "time": "2021-03-22T23:21:40.734Z",
    "resources": ["arn:aws:proton:region_id:123456789012:service-template/sample-
service-template-name:1.0"],
    "detail": {
        "name": "sample-service-template-name",
        "majorVersion": "1",
        "minorVersion": "0",
        "status": "REGISTRATION_FAILED",
        "previousStatus": "REGISTRATION_IN_PROGRESS"
    }
}
```

環境

```
{
    "source": "aws.proton",
    "detail-type": ["AWS Proton Environment Status Change"],
    "time": "2021-03-22T23:21:40.734Z",
    "resources": ["arn:aws:proton:region_id:123456789012:environment/sample-
environment"],
    "detail": {
        "name": "sample-environment",
        "status": "DELETE_FAILED",
        "previousStatus": "DELETE_IN_PROGRESS"
    }
```

```
}
```

EventBridgeTutorial: AWS Proton サービスステータスの変更に関する Amazon Simple Notification Service アラートを送信する

このチュートリアルでは、 AWS Proton サービスのステータス変更をキャプチャする AWS Proton 事前設定されたイベントルールを使用します。EventBridge は、ステータスの変更を Amazon SNS トピックに送信します。トピックをサブスクライブすると、Amazon SNS から AWS Proton サービスのステータス変更 E メールが送信されます。

前提条件

`Active` ステータスが の既存の AWS Proton サービスがあります。このチュートリアルの一部として、このサービスにサービスインスタンスを追加してからインスタンスを削除できます。

AWS Proton サービスを作成する必要がある場合は、「」を参照してください開始方法。詳細については、「AWS Proton クォータ」および「the section called “編集”」を参照してください。

ステップ 1: Amazon SNS トピックを作成してサブスクライブする

ステップ 2 で作成したイベントルールのイベントターゲットとして機能する Amazon SNS トピックを作成します。

Amazon SNS トピックを作成します。

1. ログインして Amazon SNS コンソールを開きます。
2. ナビゲーションペインで [Topics (トピック)]、[Create topic (トピックの作成)] の順に選択します。
3. [Create topic (トピックの作成)] ページで以下の操作をします。
   a. [Type (タイプ)] で [Standard (標準)] を選択します。
   b. [Name (名前)] として **`tutorial-service-status-change`** を入力し、[Create topic (トピックを作成)] を選択します。
4. tutorial-service-status-change の詳細ページで [Create subscription (サブスクリプションの作成)] を選択します。
5. [Create subscription (サブスクリプションの作成)] ページで以下の操作をします。

a. [Protocol (プロトコル)] として [Email (E メール)] を選択します。

b. [Endpoint (エンドポイント)] では、現在アクセスできるメールアドレスを入力し、[Create subscription (サブスクリプションの作成)] を選択します。

6. メールアカウントを確認し、サブスクリプションの確認メールメッセージが届くのを待ちます。確認メールが届いたら、[Confirm subscription (サブスクリプションの確認)] を選択します。

ステップ 2: イベントルールを登録する

AWS Proton サービスのステータス変更をキャプチャするイベントルールを登録します。詳細については、「前提条件」を参照してください。

イベントルールを作成します。

1. Amazon EventBridge コンソールを開きます。
2. ナビゲーションペインで、[Events (イベント)]、[Rules (ルール)] を選択します。
3. [Rules (ルール)] ページの [Rules] (ルール) セクションで [Create rules (ルールの作成)]を選択します。
4. [Create rule (ルールの作成)] ページで以下の操作をします。

   a. [Name and description (名前と説明)] セクションで [Name (名前)] として **`tutorial-rule`**を入力します。

   b. [Define pattern (パターンの定義)] セクションで [Event pattern (イベントパターン)] を選択します。

      i. [Event matching pattern (イベント照合パターン)] で、[Pre-defined by service(サービスごとの事前定義)] を選択します。

      ii. [Service provider (サービスプロバイダー)] で、「AWS」を選択します。

      iii. [Service name (サービス名)] で、AWS Proton を選択します。

      iv. [Event type (イベントタイプ)] として [AWS Proton Service Status Change (サービスステータスの変更)] を選択します。

      イベントパターンがテキストエディタに表示されます。

      v. AWS Proton コンソール を開きます。

      vi. ナビゲーションペインで [Services (サービス)] を選択します。

      vii. サービスページで、 AWS Proton サービスの名前を選択します。

viii. [Service details (サービスの詳細)] ページで、Amazon リソースネーム (ARN) サービスをコピーします。

ix. EventBridge コンソールのチュートリアルルールに戻り、テキストエディタで [Edit (編集)] を選択します。

x. テキストエディタで `"resources":` について、ステップ vii でコピーしたサービス ARN を入力します。

```
{
    "source": ["aws.proton"],
    "detail-type": ["AWS Proton Service Status Change"],
    "resources": ["arn:aws:proton:region-id:123456789012:service/your-service"]
}
```

xi. イベントパターンを保存します。

c. [Select targets (ターゲットを選択)] セクションで以下の操作をします。

i. [Target (ターゲット)] で [SNS topic (SNS トピック)] を選択します。

ii. [Topic (トピック)] として tutorial-service-status-change を選択します。

d. [Create (作成)] を選択します。

ステップ 3: イベントルールをテストする

AWS Proton サービスにインスタンスを追加して、イベントルールが機能していることを確認します。

1. AWS Proton コンソールに切り替えます。
2. ナビゲーションペインで [Services (サービス)] を選択します。
3. [Services (サービス)] ページでサービスの名前を選択します。
4. [Service details (サービスの詳細)] ページで [Edit (編集)] を選択します。
5. [Configure service (サービスの設定) ページで [Next (次へ)] を選択します。
6. [Configure custom settings (カスタム設定の構成)] ページの [Service instaces (サービスインスタンス) セクションで [Add new instance (新しいインスタンスの追加)] を選択します。
7. [New instance (新しいインスタンス)] フォームの入力を完了します。

a. 新しいインスタンスの名前を入力します。

b. 既存のインスタンスについて選択したのと同じ互換環境を選択します。

c. 必要な入力値を入力します。

d. [Next (次へ)] を選択します。

8. 入力内容を確認して [Update (更新)] を選択します。

9. サービスのステータスが になったらActive、E メールをチェックして、ステータスを更新する AWS 通知を受け取ったことを確認します。

```
{
    "version": "0",
    "id": "af76c382-2b3c-7a0a-cf01-936dff228276",
    "detail-type": "AWS Proton Service Status Change",
    "source": "aws.proton",
    "account": "123456789012",
    "time": "2021-06-29T20:40:16Z",
    "region": "region-id",
    "resources": ["arn:aws:proton:region-id:123456789012:service/your-service"],
    "detail": {
        "previousStatus": "ACTIVE",
        "status": "UPDATE_IN_PROGRESS",
        "name": "your-service"
    }
}
```

```
{
    "version": "0",
    "id": "87131e29-ad95-bda2-cd30-0ce825dfb0cd",
    "detail-type": "AWS Proton Service Status Change",
    "source": "aws.proton",
    "account": "123456789012",
    "time": "2021-06-29T20:42:27Z",
    "region": "region-id",
    "resources": ["arn:aws:proton:region-id:123456789012:service/your-service"],
    "detail": {
        "previousStatus": "UPDATE_IN_PROGRESS",
        "status": "ACTIVE",
        "name": "your-service"
    }
}
```

ステップ 4: クリーンアップする

Amazon SNS トピックとサブスクリプションを削除し、EventBridge ルールを削除します。

Amazon SNS トピックおよびサブスクリプションを削除します。

1. Amazon SNS コンソールに移動します。
2. ナビゲーションパネルで [Subscriptions (サブスクリプション)] を選択します。
3. [Subscriptions (サブスクリプション)] ページで、`tutorial-service-status-change` という名前のトピックに対して作成したサブスクリプションを選択してから [Delete (削除)] を選択します。
4. ナビゲーションペインで [Topics (トピック)] を選択します。
5. [Topics (トピック)] ページで「`tutorial-service-status-change`」という名前のトピックを選択してから [Delete (削除)] を選択します。
6. モーダルは、削除の確認を求めるプロンプトを表示します。指示に従って操作し、[Delete (削除)] を選択します。

EventBridge ルールを削除します。

1. Amazon EventBridge コンソールにサインインします。
2. ナビゲーションペインで、[Events (イベント)]、[Rules (ルール)] を選択します。
3. [Rules (ロール)] ページで `tutorial-rule`という名前のルールを選択してから [Delete (削除)] を選択します。
4. モーダルは、削除の確認を求めるプロンプトを表示します。[Delete (削除)] を選択します。

追加したサービスインスタンスを削除します。

1. AWS Proton コンソールに移動します。
2. ナビゲーションペインで [Services (サービス)] を選択します。
3. [Services (サービス)] ページで、サービスの名前を選択します。
4. [Service details (サービスの詳細)] ページで [Edit (編集)] を選択してから [Next (次へ)] を選択します。
5. [Configure custom settings (カスタム設定の構成)] ページの [Service instances (サービスインスタンス)] セクションで、このチュートリアルの一部として作成したサービスインスタンスについて [Delete (削除)] を選択してから [Next (次へ)] を選択します。

6. 入力内容を確認して [Update (更新)] を選択します。
7. モーダルは、削除の確認を求めるプロンプトを表示します。指示に従って操作し、[Yes, delete (はい、削除します)] を選択します。

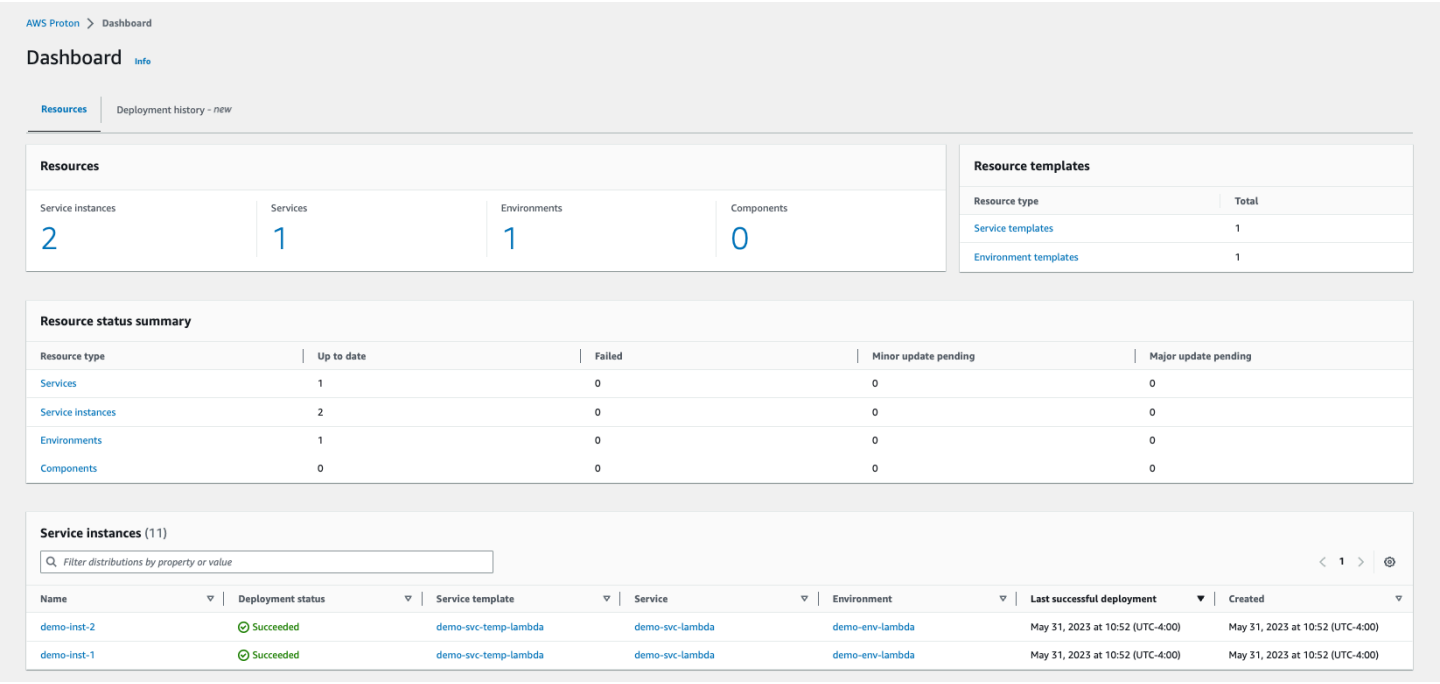
AWS Proton ダッシュボードでインフラストラクチャを最新の状態に保つ

AWS Proton ダッシュボードには、 AWS アカウントの AWS Proton リソースの概要が表示されます。特に、デプロイされたリソースの更新状況など、古さに焦点を当てています。デプロイされたリソースは、関連するテンプレートの推奨バージョンを使用すると最新の状態になります。デプロイされた古いリソースには、テンプレートのメジャーバージョンまたはマイナーバージョンの更新が必要な場合があります。

AWS Proton コンソールでダッシュボードを表示する

AWS Proton ダッシュボードを表示するには、AWS Proton コンソールを開き、ナビゲーションペインで Dashboard を選択します。

リソース



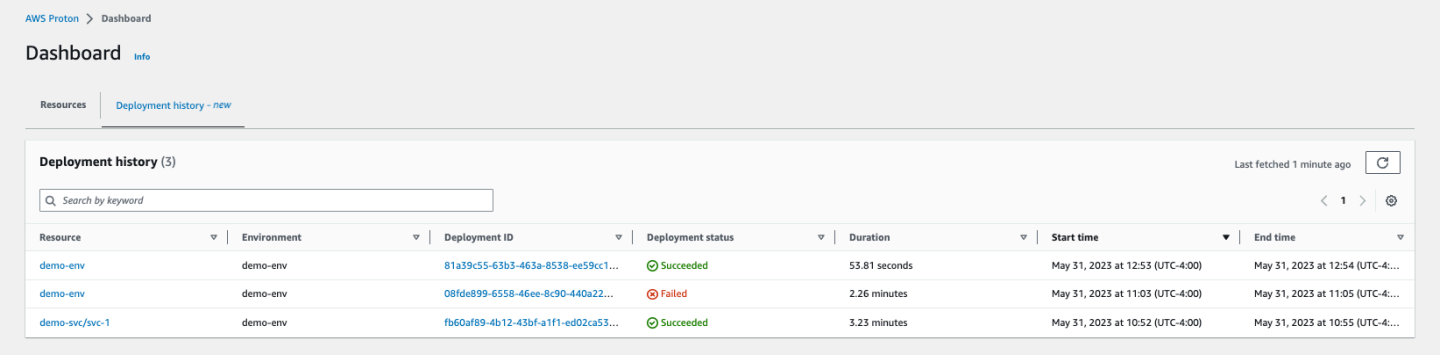
ダッシュボードの最初のタブには、アカウント内のすべてのリソースの数が表示されます。リソースタブには、サービスインスタンス、サービス、環境、コンポーネントの数と、リソーステンプレート

が表示されます。ここでは、デプロイされた各リソースタイプのリソース数が、そのタイプのリソースのステータス別に分類されます。サービスインスタンステーブルには、デプロイステータス、関連付けられている AWS Proton リソース、使用可能な更新、タイムスタンプなど、各サービスインスタンスの詳細が表示されます。

サービスインスタンスリストは、任意のテーブルプロパティでフィルタリングできます。たとえば、特定の期間内にデプロイされたサービスインスタンスや、メジャーバージョンまたはマイナーバージョンの推奨と比較して古くなっているサービスインスタンスを表示するようにフィルタリングできます。

サービスインスタンス名を選択してサービスインスタンスの詳細ページに移動し、そこで適切なバージョン更新を行うことができます。他の AWS Proton リソース名を選択して詳細ページに移動するか、リソースタイプを選択してそれぞれのリソースリストに移動します。

デプロイ履歴



[デプロイ履歴] タブでは、デプロイに関する詳細を確認できます。デプロイ履歴テーブルでは、デプロイステータス、環境、デプロイ ID を追跡できます。リソース名またはデプロイ ID を選択すると、デプロイステータスメッセージやリソース出力などの詳細を確認できます。このテーブルでは、任意のテーブルプロパティでフィルタリングすることもできます。

のセキュリティ AWS Proton

のクラウドセキュリティが最優先事項 AWS です。お客様は AWS 、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャからメリットを得られます。

セキュリティは、 AWS お客様とお客様の間の責任共有です。責任共有モデルではこれをクラウドのセキュリティおよびクラウド内のセキュリティと説明しています。

- クラウドのセキュリティ – AWS は、 AWS のサービス で実行されるインフラストラクチャを保護する責任を担います AWS クラウド。 は、お客様が安全に使用できるサービス AWS も提供します。サードパーティーの監査者は、AWS コンプライアンスプログラムコンプライアンスプログラムの一環として、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。が適用されるコンプライアンスプログラムの詳細については AWS Proton、「コンプライアンスAWS のサービス プログラムによる対象範囲内コンプライアンスプログラムによる対象範囲内」を参照してください。
- クラウド内のセキュリティ – お客様の責任は、使用する によって決まり AWS のサービス ます。また、ユーザーは、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、 を使用する際の責任共有モデルの適用方法を理解するのに役立ちます AWS Proton。以下のトピックでは、セキュリティおよびコンプライアンスの目的を達成するために AWS Proton を設定する方法を示します。また、 リソースのモニタリングと保護 AWS のサービス AWS Proton に役立つ他の の使用方法についても説明します。

トピック

- の Identity and Access Management AWS Proton
- での設定と脆弱性の分析 AWS Proton
- でのデータ保護 AWS Proton
- のインフラストラクチャセキュリティ AWS Proton
- での口グ記録とモニタリング AWS Proton
- の耐障害性 AWS Proton
- のセキュリティのベストプラクティス AWS Proton
- サービス間の混乱した代理の防止
- CodeBuild プロビジョニングカスタム Amazon VPC サポート

の Identity and Access Management AWS Proton

AWS Identity and Access Management (IAM) は、管理者が AWS リソースへのアクセスを安全に制御 AWS のサービス するのに役立つ です。IAM 管理者は、誰を認証 (サインイン) し、誰に AWS Proton リソースの使用を許可する (アクセス許可を付与する) かを制御します。IAM は、追加料金なしで使用できる AWS のサービス です。

トピック

オーディエンス

AWS Identity and Access Management (IAM) の使用方法は、ロールによって異なります。

- サービスユーザー - 機能にアクセスできない場合は、管理者にアクセス許可をリクエストします (「AWS Proton ID とアクセスのトラブルシューティング」を参照)。
- サービス管理者 - ユーザーアクセスを決定し、アクセス許可リクエストを送信します (「が IAM と AWS Proton 連携する方法」を参照)
- IAM 管理者 - アクセスを管理するためのポリシーを作成します (「のアイデンティティベースのポリシーの例 AWS Proton」を参照)

アイデンティティを使用した認証

認証とは、ID 認証情報 AWS を使用して にサインインする方法です。、IAM ユーザー AWS アカウントのルートユーザー、または IAM ロールを引き受けることで認証される必要があります。

AWS IAM アイデンティティセンター (IAM Identity Center)、シングルサインオン認証、Google/Facebook 認証情報などの ID ソースからの認証情報を使用して、フェデレーティッド ID としてサイ

ンインできます。サインインの詳細については、「AWS サインイン ユーザーガイド」の「AWS アカウントにサインインする方法」を参照してください。

プログラムによるアクセスの場合、 は SDK と CLI AWS を提供してリクエストを暗号化して署名します。詳細については、「IAM ユーザーガイド」の「API リクエストに対するAWS 署名バージョン 4」を参照してください。

AWS アカウント ルートユーザー

を作成するときは AWS アカウント、まず、すべての AWS のサービス および リソースへの完全なアクセス権を持つ AWS アカウント root ユーザーと呼ばれる 1 つのサインインアイデンティティから始めます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザー認証情報を必要とするタスクについては、「IAM ユーザーガイド」の「ルートユーザー認証情報が必要なタスク」を参照してください。

フェデレーテッドアイデンティティ

ベストプラクティスとして、人間のユーザーが一時的な認証情報 AWS のサービス を使用して にアクセスするには、ID プロバイダーとのフェデレーションを使用する必要があります。

フェデレーティッド ID は、エンタープライズディレクトリ、ウェブ ID プロバイダー、または ID ソースの認証情報 AWS のサービス を使用して Directory Service にアクセスするユーザーです。フェデレーテッドアイデンティティは、一時的な認証情報を提供するロールを引き受けます。

アクセスを一元管理する場合は、 AWS IAM アイデンティティセンターをお勧めします。詳細については、「AWS IAM アイデンティティセンター ユーザーガイド」の「IAM アイデンティティセンターとは」を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、特定の個人やアプリケーションに対する特定のアクセス許可を持つアイデンティティです。長期認証情報を持つ IAM ユーザーの代わりに一時的な認証情報を使用することをお勧めします。詳細については、IAM ユーザーガイドの「ID プロバイダーとのフェデレーションを使用して にアクセスすることを人間 AWS のユーザーに要求する」を参照してください。

IAM グループは、IAM ユーザーの集合を指定し、大量のユーザーに対するアクセス許可の管理を容易にします。詳細については、「IAM ユーザーガイド」の「IAM ユーザーに関するユースケース」を参照してください。

IAM ロール

IAM ロールは、特定のアクセス許可を持つアイデンティであり、一時的な認証情報を提供します。ユーザーから IAM ロール (コンソール) に切り替えるか、 または API オペレーションを呼び出すことで、ロールを引き受けることができます。 AWS CLI AWS 詳細については、「IAM ユーザーガイド」の「ロールを引き受けるための各種方法」を参照してください。

IAM ロールは、フェデレーションユーザーアクセス、一時的な IAM ユーザーのアクセス許可、クロスアカウントアクセス、クロスサービスアクセス、および Amazon EC2 で実行するアプリケーションに役立ちます。詳細については、IAM ユーザーガイド の IAM でのクロスアカウントリソースアクセス を参照してください。

ポリシーを使用したアクセスの管理

でアクセスを制御する AWS には、ポリシーを作成し、ID AWS またはリソースにアタッチします。ポリシーは、アイデンティティまたはリソースに関連付けられたときにアクセス許可を定義します。は、プリンシパルがリクエストを行うときにこれらのポリシー AWS を評価します。ほとんどのポリシーは JSON ドキュメント AWS として に保存されます。JSON ポリシードキュメントの詳細については、「IAM ユーザーガイド」の「JSON ポリシー概要」を参照してください。

管理者は、ポリシーを使用して、どのプリンシパルがどのリソースに対して、どのような条件でアクションを実行できるかを定義することで、誰が何にアクセスできるかを指定します。

デフォルトでは、ユーザーやロールにアクセス許可はありません。IAM 管理者は IAM ポリシーを作成してロールに追加し、このロールをユーザーが引き受けられるようにします。IAM ポリシーは、オペレーションの実行方法を問わず、アクセス許可を定義します。

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 (ユーザー、グループ、またはロール) 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。これらのポリシーは、アイデンティティがどのリソースに対してどのような条件下でどのようなアクションを実行できるかを制御します。アイデンティティベースポリシーの作成方法については、IAM ユーザーガイド の カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する を参照してください。

アイデンティティベースのポリシーは、インラインポリシー (単一の ID に直接埋め込む) または管理ポリシー (複数の ID にアタッチされたスタンドアロンポリシー) にすることができます。管理ポリシーとインラインポリシーのいずれかを選択する方法については、「IAM ユーザーガイド」の「管理ポリシーとインラインポリシーのいずれかを選択する」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。例としては、IAM ロール信頼ポリシーや Amazon S3 バケットポリシーなどがあります。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスを制御できます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーでは、IAM の AWS マネージドポリシーを使用できません。

その他のポリシータイプ

AWS は、より一般的なポリシータイプによって付与されるアクセス許可の最大数を設定できる追加のポリシータイプをサポートしています。

- アクセス許可の境界 – アイデンティティベースのポリシーで IAM エンティティに付与することのできるアクセス許可の数の上限を設定します。詳細については、「IAM ユーザーガイド」の「IAM エンティティのアクセス許可境界」を参照してください。
- サービスコントロールポリシー (SCP) - AWS Organizations内の組織または組織単位の最大のアクセス許可を指定します。詳細については、「AWS Organizations ユーザーガイド」の「サービスコントロールポリシー」を参照してください。
- リソースコントロールポリシー (RCP) – は、アカウント内のリソースで利用できる最大数のアクセス許可を定義します。詳細については、「AWS Organizations ユーザーガイド」の「リソースコントロールポリシー (RCP)」を参照してください。
- セッションポリシー – ロールまたはフェデレーションユーザーの一時セッションを作成する際にパラメータとして渡される高度なポリシーです。詳細については、「IAM ユーザーガイド」の「セッションポリシー」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成されるアクセス許可を理解するのがさらに難しくなります。が複数のポリシータイプが関与する場合にリクエストを許可するかどうか AWS を決定する方法については、「IAM ユーザーガイド」の「ポリシー評価ロジック」を参照してください。

が IAM と AWS Proton 連携する方法

IAM を使用して へのアクセスを管理する前に AWS Proton、 で使用できる IAM 機能を確認してください AWS Proton。

で使用できる IAM 機能 AWS Proton

IAM 機能	AWS Proton サポート
アイデンティティベースのポリシー	あり
リソースベースのポリシー	なし
ポリシーアクション	あり
ポリシーリソース	あり
ポリシー条件キー	あり
ACL	なし
ABAC (ポリシー内のタグ)	あり
一時的な認証情報	あり
プリンシパルアクセス権限	あり
サービスロール	あり
サービスリンクロール	はい

AWS Proton およびその他の がほとんどの IAM 機能と AWS のサービス 連携する方法の概要を把握するには、IAM ユーザーガイドのAWS のサービス 「IAM と連携する 」を参照してください。

のアイデンティティベースのポリシー AWS Proton

アイデンティティベースのポリシーのサポート: あり

アイデンティティベースポリシーは、IAM ユーザー、ユーザーグループ、ロールなど、アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティ

ベースポリシーの作成方法については、「IAM ユーザーガイド」の「カスタマー管理ポリシーでカスタム IAM アクセス許可を定義する」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。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。

のアイデンティティベースのポリシーの例 AWS Proton

AWS Proton 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のアイデンティティベースのポリシーの例 AWS Proton。

内のリソースベースのポリシー AWS Proton

リソースベースのポリシーのサポート: なし

リソースベースのポリシーは、リソースに添付する JSON ポリシードキュメントです。リソースベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげられます。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスをコントロールできます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーで、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS のサービス。

クロスアカウントアクセスを有効にするには、全体のアカウント、または別のアカウントの IAM エンティティを、リソースベースのポリシーのプリンシパルとして指定します。詳細については、IAM ユーザーガイドのIAM でのクロスアカウントリソースアクセスを参照してください。

のポリシーアクション AWS Proton

ポリシーアクションのサポート: あり

管理者は JSON AWS 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ような条件下で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。このアクションは関連付けられたオペレーションを実行するためのアクセス許可を付与するポリシーで使用されます。

AWS Proton アクションのリストを確認するには、「サービス認可リファレンス」の「 で定義されるアクション AWS Proton」を参照してください。

のポリシーアクションは、アクションの前に次のプレフィックス AWS Proton を使用します。

```
proton
```

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": [
      "proton:action1",
      "proton:action2"
         ]
```

ワイルドカード (*) を使用して複数アクションを指定できます。例えば、List という単語で始まるすべてのアクションを指定するには次のアクションを含めます。

```
"Action": "proton:List*"
```

AWS Proton 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のアイデンティティベースのポリシーの例 AWS Proton。

のポリシーリソース AWS Proton

ポリシーリソースのサポート: あり

管理者は JSON AWS 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ような条件下でアクションを実行できるかということです。

Resource JSON ポリシー要素はアクションが適用されるオブジェクトを指定します。ベストプラクティスとして、Amazon リソースネーム (ARN) を使用してリソースを指定します。リソースレベルのアクセス許可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": "*"
```

AWS Proton リソースタイプとその ARNs「 で定義されるリソース AWS Proton」を参照してください。 どのアクションで各リソースの ARN を指定できるかについては、「AWS Protonで定義されるアクション」を参照してください。

AWS Proton 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のアイデンティティベースのポリシーの例 AWS Proton。

のポリシー条件キー AWS Proton

サービス固有のポリシー条件キーのサポート: あり

管理者は JSON AWS 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ような条件下でアクションを実行できるかということです。

Condition 要素は、定義された基準に基づいてステートメントが実行される時期を指定します。イコールや未満などの条件演算子を使用して条件式を作成して、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。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。

AWS Proton 条件キーのリストを確認するには、「サービス認可リファレンス」の「 の条件キー AWS Proton」を参照してください。条件キーを使用できるアクションとリソースについては、「 で定義されるアクション AWS Proton」を参照してください。

リソースへのアクセスを制限するための条件キーベースのポリシーの例をご覧いただくには、「の条件キーベースのポリシーの例 AWS Proton」を参照してください。

のアクセスコントロールリスト (ACLs) AWS Proton

ACL のサポート: なし

アクセスコントロールリスト (ACL) は、どのプリンシパル (アカウントメンバー、ユーザー、またはロール) がリソースにアクセスするためのアクセス許可を持つかを制御します。ACL はリソースベースのポリシーに似ていますが、JSON ポリシードキュメント形式は使用しません。

アクセスコントロールリスト (ACL) は、リソースにアタッチできる被付与者のリストです。これらは、アタッチされているリソースにアクセスするための権限をアカウントに付与します。

を使用した属性ベースのアクセスコントロール (ABAC) AWS Proton

ABAC (ポリシー内のタグ) のサポート: あり

属性ベースのアクセス制御 (ABAC) は、タグと呼ばれる属性に基づいてアクセス許可を定義する認可戦略です。IAM エンティティと AWS リソースにタグをアタッチし、プリンシパルのタグがリソースのタグと一致するときにオペレーションを許可するように ABAC ポリシーを設計できます。

タグに基づいてアクセスを管理するには、aws:ResourceTag/*key-name*、aws:RequestTag/*key-name*、または aws:TagKeys の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。

サービスがすべてのリソースタイプに対して 3 つの条件キーすべてをサポートする場合、そのサービスの値はありです。サービスが一部のリソースタイプに対してのみ 3 つの条件キーのすべてをサポートする場合、値は「部分的」になります。

ABAC の詳細については、「IAM ユーザーガイド」の「ABAC 認可でアクセス許可を定義する」を参照してください。ABAC をセットアップする手順を説明するチュートリアルについては、「IAM ユーザーガイド」の「属性ベースのアクセスコントロール (ABAC) を使用する」を参照してください。

AWS Proton リソースのタグ付けの詳細については、「」を参照してくださいAWS Proton リソースとタグ付け。

での一時的な認証情報の使用 AWS Proton

一時的な認証情報のサポート: あり

一時的な認証情報は AWS 、リソースへの短期的なアクセスを提供し、フェデレーションまたは切り替えロールの使用時に自動的に作成されます。長期的なアクセスキーを使用する代わりに、一時的な認証情報を動的に生成 AWS することをお勧めします。詳細については、「IAM ユーザーガイド」の「IAM の一時的な認証情報」および「AWS のサービス と IAM との連携」を参照してください。

のクロスサービスプリンシパルのアクセス許可 AWS Proton

転送アクセスセッション (FAS) のサポート: あり

転送アクセスセッション (FAS) は、 を呼び出すプリンシパルのアクセス許可と AWS のサービス、ダウンストリームサービス AWS のサービス へのリクエストをリクエストする を使用します。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

のサービスロール AWS Proton

サービスロールのサポート: あり

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。詳細につい

ては、IAM ユーザーガイドの AWS のサービスに許可を委任するロールを作成するを参照してください。

詳細については、「AWS Proton IAM サービスロールポリシーの例」を参照してください。

> **Warning**
>
> サービスロールのアクセス許可を変更すると、 AWS Proton 機能が破損する可能性があります。 AWS Proton が指示する場合にのみ、サービスロールを編集します。

のサービスにリンクされたロール AWS Proton

サービスリンクロールのサポート: あり

サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、サービスによって所有されます。IAM 管理者は、サービスリンクロールのアクセス許可を表示できますが、編集することはできません。

詳細については、「のサービスにリンクされたロールの使用 AWS Proton」を参照してください。

のポリシーの例 AWS Proton

IAM AWS Proton ポリシーの例については、以下のセクションを参照してください。

トピック

- のアイデンティティベースのポリシーの例 AWS Proton
- AWS Proton IAM サービスロールポリシーの例
- の条件キーベースのポリシーの例 AWS Proton

のアイデンティティベースのポリシーの例 AWS Proton

デフォルトでは、 ユーザーおよびロールには、 AWS Proton リソースを作成または変更する権限はありません。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権限をユーザーに付与する IAM ポリシーを作成できます。

これらのサンプルの JSON ポリシードキュメントを使用して IAM 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 ポリシーを作成する (コンソール)」を参照してください。

各リソースタイプの ARN の形式など AWS Proton、 で定義されるアクションとリソースタイプの詳細については、「サービス認可リファレンス」の「 のアクション、リソース、および条件キー AWS Proton」を参照してください。 ARNs

トピック

- ポリシーに関するベストプラクティス
- のアイデンティティベースのポリシー例へのリンク AWS Proton

ポリシーに関するベストプラクティス

ID ベースのポリシーは、アカウント内の AWS Proton リソースを作成、アクセス、または削除できるかどうかを決定します。これらのアクションでは、 AWS アカウントに費用が発生する場合があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS 管理ポリシーを開始し、最小特権のアクセス許可に移行 – ユーザーとワークロードにアクセス許可の付与を開始するには、多くの一般的なユースケースにアクセス許可を付与するAWS 管理ポリシーを使用します。これらは で使用できます AWS アカウント。ユースケースに固有の AWS カスタマー管理ポリシーを定義することで、アクセス許可をさらに減らすことをお勧めします。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 または ジョブ機能のAWS マネージドポリシー を参照してください。
- 最小特権を適用する – IAM ポリシーでアクセス許可を設定する場合は、タスクの実行に必要な許可のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権アクセス許可とも呼ばれています。IAM を使用して許可を適用する方法の詳細については、IAM ユーザーガイド の IAM でのポリシーとアクセス許可 を参照してください。
- IAM ポリシーで条件を使用してアクセスをさらに制限する - ポリシーに条件を追加して、アクションやリソースへのアクセスを制限できます。たとえば、ポリシー条件を記述して、すべてのリクエストを SSL を使用して送信するように指定できます。条件を使用して、サービスアクションが などの特定の を通じて使用されている場合に AWS のサービス、サービスアクションへのアクセスを許可することもできます CloudFormation。詳細については、IAM ユーザーガイド の IAM JSON ポリシー要素:条件 を参照してください。
- IAM アクセスアナライザー を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する - IAM アクセスアナライザー は、新規および既存のポリシーを検証して、ポリシーが IAM ポリシー言語 (JSON) および IAM のベストプラクティスに準拠するようにします。IAM アクセスアナライザーは 100 を超えるポリシーチェックと実用的な推奨事項を提供し、安全で機能的なポリシーの

作成をサポートします。詳細については、IAM ユーザーガイド の IAM Access Analyzer でポリシーを検証する を参照してください。

- 多要素認証 (MFA) を要求する – で IAM ユーザーまたはルートユーザーを必要とするシナリオがある場合は AWS アカウント、セキュリティを強化するために MFA を有効にします。API オペレーションが呼び出されるときに MFA を必須にするには、ポリシーに MFA 条件を追加します。詳細については、IAM ユーザーガイド の MFA を使用した安全な API アクセス を参照してください。

IAM でのベストプラクティスの詳細については、IAM ユーザーガイド の IAM でのセキュリティのベストプラクティス を参照してください。

のアイデンティティベースのポリシー例へのリンク AWS Proton

のアイデンティティベースのポリシーの例へのリンク AWS Proton

- AWS の 管理ポリシー AWS Proton
- AWS Proton IAM サービスロールポリシーの例
- の条件キーベースのポリシーの例 AWS Proton

AWS Proton IAM サービスロールポリシーの例

管理者は、環境テンプレートとサービステンプレートで定義されているように が AWS Proton 作成するリソースを所有および管理します。IAM サービスロールをアカウントにアタッチ AWS Proton して、 がユーザーに代わってリソースを作成できるようにします。管理者は、 がアプリケーションを AWS Proton サービスとして AWS Proton 環境に AWS Proton デプロイするときに、開発者が後で所有および管理しているリソースの IAM ロールと AWS Key Management Service キーを提供します。 AWS KMS およびデータ暗号化の詳細については、「」を参照してくださいでのデータ保護 AWS Proton。

サービスロールは、 がユーザーに代わって リソースを呼び出す AWS Proton ことができる Amazon Web Services (IAM) ロールです。サービスロールを指定する場合、 AWS Proton はロールの認証情報を使用します。サービスロールを使用して、 AWS Proton が実行できるアクションを明示的に指定します。

IAM サービスで、サービスロールと権限ポリシーを作成します。サービスロールの作成の詳細については、IAM ユーザーガイドの「 AWS サービスにアクセス許可を委任するロールの作成」を参照してください。

AWS Proton を使用してプロビジョニングするための サービスロール CloudFormation

プラットフォームチームのメンバーは、管理者として AWS Proton サービスロールを作成し、環境を環境の CloudFormation サービスロール (CreateEnvironment API アクションの `protonServiceRoleArn`パラメータ) として作成 AWS Proton するときに に提供できます。このロールにより AWS Proton 、 は、環境またはそこで実行されているサービスインスタンスが AWS マネージドプロビジョニングを使用する場合に、ユーザーに代わって他の のサービスを API コールし、インフラストラクチャ AWS CloudFormation をプロビジョニングできます。

AWS Proton サービスロールには、次の IAM ロールと信頼ポリシーを使用することをお勧めします。 AWS Proton コンソールを使用して環境を作成し、新しいロールを作成する場合、これは が作成するサービスロール AWS Proton に追加するポリシーです。このポリシーのアクセス許可をスコープダウンする場合は、 が`Access Denied`エラーで AWS Proton 失敗することに注意してください。

> **Important**
>
> 次の例に示すポリシーは、テンプレートをアカウントに登録できるすべてのユーザーに管理者権限を付与するので注意してください。 AWS Proton テンプレートで定義するリソースがわからないため、これらのポリシーには広範なアクセス許可があります。したがって、環境にデプロイする特定のリソースに対する権限は、範囲を絞り込むことをお勧めします。

AWS Proton のサービスロールポリシーの例 CloudFormation

を AWS アカウント ID *123456789012*に置き換えます。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudformation:CancelUpdateStack",
        "cloudformation:ContinueUpdateRollback",
        "cloudformation:CreateChangeSet",
        "cloudformation:CreateStack",
        "cloudformation:DeleteChangeSet",
```

```
        "cloudformation:DeleteStack",
        "cloudformation:DescribeChangeSet",
        "cloudformation:DescribeStackDriftDetectionStatus",
        "cloudformation:DescribeStackEvents",
        "cloudformation:DescribeStackResourceDrifts",
        "cloudformation:DescribeStacks",
        "cloudformation:DetectStackResourceDrift",
        "cloudformation:ExecuteChangeSet",
        "cloudformation:ListChangeSets",
        "cloudformation:ListStackResources",
        "cloudformation:UpdateStack"
      ],
      "Resource": "arn:aws:cloudformation:*:123456789012:stack/AWSProton-*"
    },
    {
      "Effect": "Allow",
      "NotAction": [
        "organizations:*",
        "account:*"
      ],
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws:CalledVia": [
            "cloudformation.amazonaws.com"
          ]
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "organizations:DescribeOrganization",
        "account:ListRegions"
      ],
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws:CalledVia": [
            "cloudformation.amazonaws.com"
          ]
        }
      }
    }
```

```
  ]
}
```

AWS Proton サービス信頼ポリシー

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Sid": "ServiceTrustRelationshipWithConfusedDeputyPrevention",
    "Effect": "Allow",
    "Principal": {
      "Service": "proton.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      },
      "ArnLike": {
        "aws:SourceArn": "arn:aws:proton:*:123456789012:environment/*"
      }
    }
  }
}
```

スコープ AWSダウンマネージドプロビジョニングサービスロールポリシー

以下は、S3 リソースのプロビジョニングに AWS Proton サービスのみが必要な場合に使用できる、スコープダウンされた AWS Proton サービスロールポリシーの例です。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
      "Action": [
        "cloudformation:CancelUpdateStack",
        "cloudformation:ContinueUpdateRollback",
        "cloudformation:CreateChangeSet",
        "cloudformation:CreateStack",
        "cloudformation:DeleteChangeSet",
        "cloudformation:DeleteStack",
        "cloudformation:DescribeChangeSet",
        "cloudformation:DescribeStackDriftDetectionStatus",
        "cloudformation:DescribeStackEvents",
        "cloudformation:DescribeStackResourceDrifts",
        "cloudformation:DescribeStacks",
        "cloudformation:DetectStackResourceDrift",
        "cloudformation:ExecuteChangeSet",
        "cloudformation:ListChangeSets",
        "cloudformation:ListStackResources",
        "cloudformation:UpdateStack"
      ],
      "Resource": "arn:aws:cloudformation:*:123456789012:stack/AWSProton-*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:*"
      ],
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws:CalledVia": [
            "cloudformation.amazonaws.com"
          ]
        }
      }
    }
  ]
}
```

AWS Proton CodeBuild プロビジョニングのサービスロール

プラットフォームチームのメンバーは、管理者として AWS Proton サービスロールを作成し、環境を環境の CodeBuild サービスロール (CreateEnvironment API アクションの `codebuildRoleArn`パラメータ) として作成 AWS Proton するときに に提供できます。このロールは AWS Proton 、環境ま

たはそこで実行されているサービスインスタンスが CodeBuild プロビジョニングを使用してインフラストラクチャをプロビジョニングするときに、 がユーザーに代わって他の サービスを API コールすることを許可します。

AWS Proton コンソールを使用して環境を作成し、新しいロールを作成する場合、 は、作成したサービスロールに管理者権限を持つポリシー AWS Proton を追加します。独自のロールを作成し、アクセス許可の範囲を絞り込むときは、 `Access Denied` エラーで が AWS Proton 失敗することに注意してください。

> **Important**
>
> が作成するロールに AWS Proton アタッチするポリシーは、テンプレートをアカウントに登録できるすべてのユーザーに管理者権限を付与することに注意してください。 AWS Proton テンプレートで定義するリソースがわからないため、これらのポリシーには広範なアクセス許可があります。したがって、環境にデプロイする特定のリソースに対する権限は、範囲を絞り込むことをお勧めします。

AWS Proton CodeBuild のサービスロールポリシーの例

次の例では、CodeBuild が AWS Cloud Development Kit (AWS CDK)でリソースをプロビジョニングするための権限のあなたは、管理者として サービスロールを作成しますが、そのロールは環境を提供します。

を AWS アカウント ID *123456789012*に置き換えます。

JSON

```
{
 "Version":"2012-10-17",
 "Statement": [
   {
     "Action": [
       "logs:CreateLogStream",
       "logs:CreateLogGroup",
       "logs:PutLogEvents"
     ],
     "Resource": [
       "arn:aws:logs:us-east-1:123456789012:log-group:/aws/codebuild/AWSProton-
Shell-*",
```

```
        "arn:aws:logs:us-east-1:123456789012:log-group:/aws/codebuild/AWSProton-
  Shell-*:*"
      ],
      "Effect": "Allow"
    },
    {
      "Action": "proton:NotifyResourceDeploymentStatusChange",
      "Resource": "arn:aws:proton:us-east-1:123456789012:*",
      "Effect": "Allow"
    },
    {
      "Action": "sts:AssumeRole",
      "Resource": [
        "arn:aws:iam::123456789012:role/cdk-*-deploy-role-*",
        "arn:aws:iam::123456789012:role/cdk-*-file-publishing-role-*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

AWS Proton CodeBuild 信頼ポリシー

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Sid": "CodeBuildTrustRelationshipWithConfusedDeputyPrevention",
    "Effect": "Allow",
    "Principal": {
      "Service": "codebuild.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      },
      "ArnLike": {
        "aws:SourceArn": "arn:aws:proton:*:123456789012:environment/*"
      }
    }
```

```
    }
}
```

AWS Proton パイプラインサービスロール

サービスパイプラインをプロビジョニングするには、 に他の サービスへの API コールを行うためのアクセス許可 AWS Proton が必要です。必要なサービスロールは、環境の作成時に指定するサービスロールと似ています。ただし、パイプラインを作成するためのロールは AWS アカウント内のすべてのサービス間で共有され、これらのロールを コンソールで、または UpdateAccountSettings API アクションを通じてアカウント設定として指定します。

AWS Proton コンソールを使用してアカウント設定を更新し、 CloudFormation または CodeBuild サービスロールのいずれかの新しいロールを作成する場合、 が作成するサービスロール AWS Proton に追加するポリシーは、前のセクションで説明したポリシー、AWSマネージドプロビジョニングロールおよび と同じですCodeBuild プロビジョニングロール。このポリシーのアクセス許可をスコープダウンする場合、 はAccess Denied エラーで AWS Proton 失敗することに注意してください。

> **Important**
>
> 前のセクションのポリシー例では、あなたのアカウントにテンプレートを登録できるすべての人に管理者権限が与えられるので注意してください。 AWS Proton テンプレートで定義するリソースがわからないため、これらのポリシーには広範なアクセス許可があります。したがって、パイプラインにデプロイする特定のリソースに対する権限は、範囲を絞り込むことをお勧めします。

AWS Proton コンポーネントロール

プラットフォームチームのメンバーは、管理者として AWS Proton サービスロールを作成し、環境を環境の CloudFormation コンポーネントロール (CreateEnvironment API アクションの componentRoleArn パラメータ) として作成 AWS Proton するときに に提供できます。このロールは、直接定義したコンポーネントがプロビジョニングできるインフラストラクチャの範囲を絞り込みます。コンポーネントの詳細については、「コンポーネント」を参照してください。

以下のポリシー例では、Amazon Simple Storage Service (Amazon S3) バケットと関連するアクセスポリシーをプロビジョニングする、直接定義コンポーネントを作成することをサポートします。

AWS Proton コンポーネントロールポリシーの例

を AWS アカウント ID *123456789012*に置き換えます。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudformation:CancelUpdateStack",
        "cloudformation:CreateChangeSet",
        "cloudformation:DeleteChangeSet",
        "cloudformation:DescribeStacks",
        "cloudformation:ContinueUpdateRollback",
        "cloudformation:DetectStackResourceDrift",
        "cloudformation:DescribeStackResourceDrifts",
        "cloudformation:DescribeStackEvents",
        "cloudformation:CreateStack",
        "cloudformation:DeleteStack",
        "cloudformation:UpdateStack",
        "cloudformation:DescribeChangeSet",
        "cloudformation:ExecuteChangeSet",
        "cloudformation:ListChangeSets",
        "cloudformation:ListStackResources"
      ],
      "Resource": "arn:aws:cloudformation:*:123456789012:stack/AWSProton-*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:CreateBucket",
        "s3:DeleteBucket",
        "s3:GetBucket*",
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:GetPolicy",
        "iam:ListPolicyVersions",
        "iam:DeletePolicyVersion"
      ],
      "Resource": "*",
```

```
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws:CalledVia": "cloudformation.amazonaws.com"
        }
      }
    }
  ]
}
```

AWS Proton コンポーネントの信頼ポリシー

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Sid": "ServiceTrustRelationshipWithConfusedDeputyPrevention",
    "Effect": "Allow",
    "Principal": {
      "Service": "proton.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      },
      "ArnLike": {
        "aws:SourceArn": "arn:aws:proton:*:123456789012:environment/*"
      }
    }
  }
}
```

の条件キーベースのポリシーの例 AWS Proton

次の IAM ポリシーの例では、 `Condition`ブロックで指定されたテンプレートに一致する AWS Proton アクションへのアクセスを拒否します。これらの条件キーは、「アクション、リソース、および条件キー AWS Proton」に記載されているアクションでのみサポートされることに注意してくだ

さい。DeleteEnvironmentTemplate など、その他のアクションに対する権限を管理するには、リソースレベルのアクセス制御を使用する必要があります。

特定の AWS Proton テンプレートのテンプレートアクションを拒否するポリシーの例:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": ["proton:*"],
            "Resource": "*",
            "Condition": {
                "StringEqualsIfExists": {
                    "proton:EnvironmentTemplate":
 ["arn:aws:proton:region_id:123456789012:environment-template/my-environment-
template"]
                }
            }
        },
        {
            "Effect": "Deny",
            "Action": ["proton:*"],
            "Resource": "*",
            "Condition": {
                "StringEqualsIfExists": {
                    "proton:ServiceTemplate":
 ["arn:aws:proton:region_id:123456789012:service-template/my-service-template"]
                }
            }
        }
    ]
}
```

次のポリシー例では、最初のリソースレベルのステートメントは、 Resource ブロックにリストされているサービス AWS Proton テンプレートに一致する ListServiceTemplates以外のテンプレートアクションへのアクセスを拒否します。2 番目のステートメントは、 Conditionブロックにリストされているテンプレートに一致する AWS Proton アクションへのアクセスを拒否します。

特定のテンプレートに一致する AWS Proton アクションを拒否するポリシーの例:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "proton:*"
            ],
            "Resource": "arn:aws:proton:us-east-1:123456789012:service-template/
my-service-template"
        },
        {
            "Effect": "Deny",
            "Action": [
                "proton:*"
            ],
            "Resource": "*",
            "Condition": {
                "StringEqualsIfExists": {
                    "proton:ServiceTemplate": [
                        "arn:aws:proton:us-east-1:123456789012:service-template/
my-service-template"
                    ]
                }
            }
        }
    ]
}
```

最後のポリシー例では、 Conditionブロックにリストされている特定のサービステンプレートに一致する開発者 AWS Proton アクションを許可します。

特定のテンプレートに一致する AWS Proton 開発者アクションを許可するポリシーの例:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "proton:ListServiceTemplates",
                "proton:ListServiceTemplateVersions",
                "proton:ListServices",
                "proton:ListServiceInstances",
                "proton:ListEnvironments",
                "proton:GetServiceTemplate",
                "proton:GetServiceTemplateVersion",
                "proton:GetService",
                "proton:GetServiceInstance",
                "proton:GetEnvironment",
                "proton:CreateService",
                "proton:UpdateService",
                "proton:UpdateServiceInstance",
                "proton:UpdateServicePipeline",
                "proton:DeleteService",
                "codestar-connections:ListConnections"
            ],
            "Resource": "*",
            "Condition": {
                "StringEqualsIfExists": {
                    "proton:ServiceTemplate":
 "arn:aws:proton:region_id:123456789012:service-template/my-service-template"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "codestar-connections:PassConnection"
            ],
            "Resource": "arn:aws:codestar-connections:*:*:connection/*",
            "Condition": {
                "StringEquals": {
                    "codestar-connections:PassedToService":
 "proton.amazonaws.com"
```

```
                }
            }
        }

    ]
}
```

AWS の 管理ポリシー AWS Proton

ユーザー、グループ、ロールにアクセス許可を追加するには、自分でポリシーを記述するよりも AWS 管理ポリシーを使用する方が簡単です。チームに必要な権限のみを提供する IAM カスタマーマネージドポリシーを作成するには時間と専門知識が必要です。すぐに開始するには、 AWS マネージドポリシーを使用できます。これらのポリシーは、一般的なユースケースをターゲット範囲に含めており、 AWS アカウントで利用できます。 AWS 管理ポリシーの詳細については、IAM ユーザーガイドの「 AWS 管理ポリシー」を参照してください。

AWS のサービス AWS 管理ポリシーを維持および更新します。 AWS 管理ポリシーのアクセス許可は変更できません。サービスでは新しい機能を利用できるようにするために、 AWS マネージドポリシーに権限が追加されることがあります。この種類の更新はポリシーがアタッチされている、すべてのアイデンティティ (ユーザー、グループおよびロール) に影響を与えます。新しい機能が立ち上げられた場合や、新しいオペレーションが使用可能になった場合に、各サービスが 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。サービスは AWS 管理ポリシーからアクセス許可を削除しないため、ポリシーの更新によって既存のアクセス許可が損なわれることはありません。

さらに、 は、複数のサービスにまたがるジョブ関数の マネージドポリシー AWS をサポートしています。例えば、ReadOnlyAccess AWS 管理ポリシーは、すべての AWS のサービス および リソースへの読み取り専用アクセスを提供します。サービスが新しい機能を起動する場合、 AWS は新たなオペレーションとリソース用に、読み取り専用の許可を追加します。ジョブ機能のポリシーの一覧および詳細については、「IAM ユーザーガイド」の「AWS のジョブ機能のマネージドポリシー」を参照してください。

AWS Proton は、リソースと API オペレーションをさまざまなレベルで制御できるユーザー、グループ、またはロールにアタッチできるマネージド IAM ポリシーと信頼関係を提供します。これらのポリシーを直接適用することも、独自のポリシーを作成する開始点として使用することもできます。

AWS Proton 管理ポリシーごとに次の信頼関係が使用されます。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": {
    "Sid": "ExampleTrustRelationshipWithProtonConfusedDeputyPrevention",
    "Effect": "Allow",
    "Principal": {
      "Service": "proton.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      },
      "ArnLike": {
        "aws:SourceArn": "arn:aws:proton:*:123456789012:environment/*"
      }
    }
  }
}
```

AWS マネージドポリシー: AWSProtonFullAccess

IAM エンティティ AWSProtonFullAccess に をアタッチできます。 は、ユーザーに代わって がアクションを実行できるようにするサービスロール AWS Proton にもこのポリシー AWS Proton をアタッチします。

このポリシーは、 AWS Proton アクションへのフルアクセスと、 AWS Proton が依存する他の AWS サービスアクションへの制限付きアクセスを許可する管理アクセス許可を付与します。

このポリシーには、以下の主要なアクション名前空間が含まれています。

- proton — 管理者に AWS Proton API へのフルアクセスを許可します。
- iam — 管理者が AWS Protonにロールを渡せるようになります。これは、 が管理者に代わって他の サービスに API コール AWS Proton を実行できるようにするために必要です。
- kms — 管理者にカスタマーマネージドキーへの権限の追加を許可します。
- codeconnections – 管理者がコード接続を一覧表示して渡して使用できるようにします AWS Proton。

詳細については、「AWSProtonFullAccess」を参照してください。

AWS マネージドポリシー: AWSProtonDeveloperAccess

IAM エンティティ AWSProtonDeveloperAccessに をアタッチできます。 は、ユーザーに代わって がアクションを実行できるようにするサービスロール AWS Proton にもこのポリシー AWS Proton をアタッチします。

このポリシーは、 AWS Proton アクションおよび AWS Proton が依存する他の AWS アクションへの制限付きアクセスを許可するアクセス許可を付与します。これらのアクセス許可の範囲は、 AWS Proton サービスを作成およびデプロイする開発者のロールをサポートするように設計されています。

このポリシーは、 AWS Proton テンプレートと環境の作成、削除、更新 APIs へのアクセスを提供しません。このポリシーで提供される権限よりもさらに制限された権限を開発者が必要とする場合は、最小特権を認める範囲まで絞ったカスタムポリシーの作成をお勧めします。

このポリシーには、以下の主要なアクション名前空間が含まれています。

- proton — コントリビューターによる AWS Proton API セットへの限定アクセス権限を認めます。
- codeconnections – コントリビューターがコード接続を一覧表示して渡して使用できるようにします AWS Proton。

詳細については、「AWSProtonDeveloperAccess」を参照してください。

AWS マネージドポリシー: AWSProtonReadOnlyAccess

IAM エンティティ AWSProtonReadOnlyAccessに をアタッチできます。 は、ユーザーに代わってがアクションを実行できるようにするサービスロール AWS Proton にもこのポリシー AWS Proton をアタッチします。

このポリシーは、 AWS Proton アクションへの読み取り専用アクセスと、 AWS Proton が依存する他の AWS サービスアクションへの読み取り専用アクセスの制限を許可するアクセス許可を付与します。

このポリシーには次の権限が含まれます。

- proton — 共同作成者に AWS Proton API への読み取り専用アクセスを許可します。

詳細については、「AWSProtonReadOnlyAccess」を参照してください。

AWS マネージドポリシー: AWSProtonSyncServiceRolePolicy

AWS Proton は、 がテンプレート同期を実行できるようにする AWSServiceRoleForProtonSync サービスにリンクされたロール AWS Proton にこのポリシーをアタッチします。

このポリシーは、 AWS Proton アクションおよび AWS Proton に依存する他の AWS サービスアクションへの制限付きアクセスを許可するアクセス許可を付与します。

このポリシーには、以下の主要なアクション名前空間が含まれています。

- `proton` – APIs への AWS Proton AWS Proton 同期制限付きアクセスを許可します。
- `codeconnections` – CodeConnections APIs への AWS Proton 同期制限付きアクセスを許可します。

詳細については、「AWSProtonSyncServiceRolePolicy」を参照してください。

AWS マネージドポリシー: AWSProtonCodeBuildProvisioningBasicAccess

アクセス許可 CodeBuild は、 AWS Proton CodeBuild プロビジョニング用のビルドを実行する必要があります。`AWSProtonCodeBuildProvisioningBasicAccess` を CodeBuild プロビジョニングロールにアタッチできます。

このポリシーは、 AWS Proton CodeBuild プロビジョニングが機能するための最小限のアクセス許可を付与します。このポリシーでは、CodeBuild がビルドログを生成するための権限が与えられます。また、 Proton が Infrastructure as Code (IaC) 出力を AWS Proton ユーザーが使用できるようにするアクセス許可も付与します。このポリシーでは、IaC ツールがインフラストラクチャを管理するのに必要な権限は与えられません。

このポリシーには、以下の主要なアクション名前空間が含まれます。

- `logs`-CodeBuild によるビルドログの生成を許可します。この権限がないと、CodeBuild は開始できません。
- `proton` - CodeBuild プロビジョニングコマンドが を呼び出して、特定の AWS Proton リソースの IaaC 出力`aws proton notify-resource-deployment-status-change`を更新できるようにします。

詳細については、「AWSProtonCodeBuildProvisioningBasicAccess」を参照してください。

AWS マネージドポリシー: AWSProtonCodeBuildProvisioningServiceRolePolicy

AWS Proton は、CodeBuild ベースのプロビジョニングの実行を に許可する AWSServiceRoleForProtonCodeBuildProvisioning サービスにリンクされたロール AWS Proton にこのポリシーをアタッチします。

このポリシーは、 AWS Proton が依存する AWS サービスアクションへの制限付きアクセスを許可するアクセス許可を付与します。

このポリシーには、以下の主要なアクション名前空間が含まれています。

- `cloudformation` – Allow AWS Proton CodeBuild ベースのプロビジョニングによる CloudFormation APIs への制限付きアクセス。
- `codebuild` – AWS Proton CodeBuild ベースのプロビジョニングによる CodeBuild APIs への制限付きアクセスを許可します。
- `iam` — 管理者が AWS Protonにロールを渡せるようになります。これは、 が管理者に代わって他の サービスに API コール AWS Proton を実行できるようにするために必要です。
- `servicequotas` – AWS Proton が CodeBuild の同時ビルド制限をチェックできるようにします。これにより、適切なビルドキューイングが保証されます。

詳細については、「AWSProtonCodeBuildProvisioningServiceRolePolicy」を参照してください。

AWS マネージドポリシー: AWSProtonServiceGitSyncServiceRolePolicy

AWS Proton は、サービス同期の実行を に許可する AWSServiceRoleForProtonServiceSync サービスにリンクされたロール AWS Proton にこのポリシーをアタッチします。

このポリシーは、 AWS Proton アクションおよび AWS Proton に依存する他の AWS サービスアクションへの制限付きアクセスを許可するアクセス許可を付与します。

このポリシーには、以下の主要なアクション名前空間が含まれています。

- `proton` – AWS Proton APIs への AWS Proton 同期制限付きアクセスを許可します。

詳細については、「AWSProtonServiceGitSyncServiceRolePolicy」を参照してください。

AWS Proton AWS 管理ポリシーの更新

このサービスがこれらの変更の追跡を開始 AWS Proton してからの の AWS 管理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アラートについては、 AWS Proton ドキュメント履歴ページの RSS フィードにサブスクライブしてください。

変更	説明	日付
AWSProtonCodeBuildProvisioningServiceRolePolicy - 既存ポリシーへの更新	が CodeBuild ベースのプロビジョニングを実行できるようにするサービスにリンクされたロールの マネージドポリシーは AWS Proton 、CloudFormation `TagResource` および `UntagResource` API アクションを呼び出すアクセス許可を付与するようになりました。これらのアクセス許可は、 リソースでタグ付けオペレーションを実行するために必要です。	2024 年 6 月 15 日
AWSProtonFullAccess – 既存ポリシーへの更新	Git リポジトリとの Git 同期を使用するためのサービスにリンクされたロールの マネージドポリシーが、両方のサービスプレフィックスを持つリソース用に更新されました。詳細については、「AWS CodeConnections のサービスにリンクされたロールの使用」および「 マネージドポリシー」を参照してください。	2024 年 4 月 25 日
AWSProtonDeveloperAccess – 既存ポリシーへの更新	Git リポジトリとの Git 同期を使用するためのサービスにリンクされたロールの マ	2024 年 4 月 25 日

変更	説明	日付
	ネージドポリシーが、両方のサービスプレフィックスを持つリソース用に更新されました。詳細については、「AWS CodeConnections のサービスにリンクされたロールの使用」および「 マネージドポリシー」を参照してください。	
AWSProtonSyncServiceRolePolicy – 既存ポリシーへの更新	Git リポジトリとの Git 同期を使用するためのサービスにリンクされたロールの マネージドポリシーが、両方のサービスプレフィックスを持つリソース用に更新されました。詳細については、「AWS CodeConnections のサービスにリンクされたロールの使用」および「 マネージドポリシー」を参照してください。	2024 年 4 月 25 日
AWSProtonCodeBuildProvisioningServiceRolePolicy – 既存ポリシーへの更新	AWS Proton はこのポリシーを更新し、CodeBuild プロビジョニングを使用するために必要な CodeBuild 同時ビルド制限がアカウントにあることを確認するアクセス許可を追加しました。	2023 年 5 月 12 日

変更	説明	日付
AWSProtonServiceGitSyncServiceRolePolicy - 新しいポリシー	AWS Proton にサービス同期 AWS Proton の実行を許可する新しいポリシーが追加されました。このポリシーは AWSServiceRoleForProtonServiceSync サービスリンクロールで使用します。	2023 年 3 月 31 日
AWSProtonDeveloperAccess – 既存ポリシーへの更新	AWS Proton に、テンプレート、デプロイされたテンプレートリソース、古いリソースの概要を表示できる新しい `GetResourcesSummary` アクションが追加されました。	2022 年 11 月 18 日
AWSProtonReadOnlyAccess – 既存ポリシーへの更新	AWS Proton に、テンプレート、デプロイされたテンプレートリソース、古いリソースの概要を表示できる新しい `GetResourcesSummary` アクションが追加されました。	2022 年 11 月 18 日
AWSProtonCodeBuildProvisioningBasicAccess - 新しいポリシー	AWS Proton は、CodeBuild プロビジョニング用のビルドを実行するために必要なアクセス許可を AWS Proton CodeBuild に付与する新しいポリシーを追加しました。	2022 年 11 月 16 日

変更	説明	日付
AWSProtonSyncServiceRolePolicy - 新しいポリシー	AWS Proton に CodeBuild ベースのプロビジョニングに関連するオペレーション AWS Proton の実行を許可する新しいポリシーが追加されました。このポリシーは AWSServiceRoleForProtonCodeBuildProvisioning サービスリンクロールで使用します。	2022 年 9 月 2 日
AWSProtonFullAccess – 既存ポリシーへの更新	AWS Proton は、新しい AWS Proton API オペレーションへのアクセスを提供し、一部の AWS Proton コンソールオペレーションのアクセス許可の問題を修正するために、このポリシーを更新しました。	2022 年 3 月 30 日
AWSProtonDeveloperAccess – 既存ポリシーへの更新	AWS Proton このポリシーを更新して、新しい AWS Proton API オペレーションへのアクセスを提供し、一部の AWS Proton コンソールオペレーションのアクセス許可の問題を修正します。	2022 年 3 月 30 日
AWSProtonReadOnlyAccess – 既存ポリシーへの更新	AWS Proton このポリシーを更新して、新しい AWS Proton API オペレーションへのアクセスを提供し、一部の AWS Proton コンソールオペレーションのアクセス許可の問題を修正します。	2022 年 3 月 30 日

変更	説明	日付
AWSProtonSyncServiceRolePolicy - 新しいポリシー	AWS Proton にテンプレート同期に関連するオペレーション AWS Proton の実行を許可する新しいポリシーが追加されました。このポリシーは AWSServiceRoleForProtonSync サービスリンクロールで使用されます。	2021 年 11 月 23 日
AWSProtonFullAccess - 新しいポリシー	AWS Proton は、 AWS Proton API オペレーションと AWS Proton コンソールへの管理ロールアクセスを提供する新しいポリシーを追加しました。	2021 年 6 月 9 日
AWSProtonDeveloperAccess - 新しいポリシー	AWS Proton は、開発者ロールに AWS Proton API オペレーションと AWS Proton コンソールへのアクセスを提供する新しいポリシーを追加しました。	2021 年 6 月 9 日
AWSProtonReadOnlyAccess - 新しいポリシー	AWS Proton AWS Proton API オペレーションと AWS Proton コンソールへの読み取り専用アクセスを提供する新しいポリシーが追加されました。	2021 年 6 月 9 日
AWS Proton は変更の追跡を開始しました。	AWS Proton は、 AWS 管理ポリシーの変更の追跡を開始しました。	2021 年 6 月 9 日

のサービスにリンクされたロールの使用 AWS Proton

AWS Proton 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、直接リンクされた一意のタイプの IAM ロールです AWS Proton。サービスにリンクされたロールは によって事前定義 AWS Proton されており、サービスがユーザーに代わって他の AWS サービスを呼び出すために必要なすべてのアクセス許可が含まれています。

トピック

- AWS Proton 同期にロールを使用する
- CodeBuild ベースのプロビジョニングにロールを使用する

AWS Proton 同期にロールを使用する

AWS Proton 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、直接リンクされた一意のタイプの IAM ロールです AWS Proton。サービスにリンクされたロールは によって事前定義 AWS Proton されており、サービスがユーザーに代わって他の AWS サービスを呼び出すために必要なすべてのアクセス許可が含まれています。

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 AWS Proton 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 AWS Proton を定義します。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 AWS Proton ることができます。定義される許可は信頼ポリシーと許可ポリシーに含まれており、その許可ポリシーを他の 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を削除するには、最初に関連リソースを削除する必要があります。これにより、 AWS Proton リソースへのアクセス許可が誤って削除されないため、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の詳細については、AWS 「IAM と連携するサービス」を参照し、「サービスにリンクされたロール」列で「はい」を持つサービスを探します。サービスリンクロールに関するドキュメントをサービスで表示するには、リンクで [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 AWS Proton

AWS Proton は、AWSServiceRoleForProtonSync と AWSServiceRoleForProtonServiceSync という 2 つのサービスにリンクされたロールを使用します。

AWSServiceRoleForProtonSync サービスにリンクされたロールは、次のサービスを信頼してロールを引き受けます。

- `sync.proton.amazonaws.com`

という名前のロールアクセス許可ポリシー`AWSProtonSyncServiceRolePolicy`により AWS Proton 、 は指定されたリソースに対して次のアクションを実行できます。

- アクション:AWS Proton テンプレートとテンプレートバージョンの作成、管理、読み取り
- アクション: CodeConnections で接続を使用する

このポリシーの詳細については、「AWS マネージドポリシー: AWSProtonSyncServiceRolePolicy」を参照してください。

AWSServiceRoleForProtonServiceSync サービスにリンクされたロールは、次のサービスを信頼してロールを引き受けます。

- `service-sync.proton.amazonaws.com`

という名前のロールアクセス許可ポリシー`AWSProtonServiceGitSyncServiceRolePolicy`により AWS Proton 、 は指定されたリソースに対して次のアクションを実行できます。

- アクション: AWS Proton サービスおよびサービスインスタンスでの作成、管理、読み取り

このポリシーの詳細については、「AWS マネージドポリシー: AWSProtonServiceGitSyncServiceRolePolicy」を参照してください。

サービスリンクロールの作成、編集、削除を IAM エンティティ (ユーザー、グループ、ロールなど) に許可するにはアクセス許可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「サービスにリンクされた役割のアクセス許可」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 AWS Proton

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。、、または AWS API AWS Proton で同期するようにリポジトリ AWS マネジメントコンソール AWS CLIまたはサービスを設定すると、AWS Proton によってサービスにリンクされたロールが作成されます。

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。同期用にリポジトリまたはサービスを設定すると AWS Proton、 はサービスにリンクされたロールを再度 AWS Proton 作成します。

AWSServiceRoleForProtonSync サービスにリンクされたロールを再作成するには、同期用にリポジトリを設定し、AWSServiceRoleForProtonServiceSync を再作成するには、同期用にサービスを設定します。

のサービスにリンクされたロールの編集 AWS Proton

AWS Proton では、AWSServiceRoleForProtonSync 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 を使用したロール記述の編集はできます。詳細については、「IAM ユーザーガイド」の「サービスリンクロールの編集」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 AWS Proton

AWSServiceRoleForProtonSync ロールを手動で削除する必要はありません。 AWS マネジメントコンソール、、または AWS API でリポジトリ同期用の AWS Proton リンクされたリポジトリをすべて削除すると、 AWS Proton はリソースをクリーンアップし AWS CLI、サービスにリンクされたロールを削除します。

AWS Proton サービスにリンクされたロールでサポートされているリージョン

AWS Proton は、サービス AWS リージョン が利用可能なすべての 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 全般のリファレンスの「AWS Proton エンドポイントとクォータ」を参照してください。

CodeBuild ベースのプロビジョニングにロールを使用する

AWS Proton 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、直接リンクされた一意のタイプの IAM ロールです AWS Proton。サービスにリンクされたロールは によって事前定義 AWS Proton されており、サービスがユーザーに代わって他の AWS サービスを呼び出すために必要なすべてのアクセス許可が含まれています。

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 AWS Proton 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 AWS Proton を定義します。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 AWS

Proton ることができます。定義される許可は信頼ポリシーと許可ポリシーに含まれており、その許可ポリシーを他の 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を削除するには、最初に関連リソースを削除する必要があります。これにより、 AWS Proton リソースへのアクセス許可が誤って削除されないため、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の詳細については、AWS 「IAM と連携するサービス」を参照し、「サービスにリンクされたロール」列で「はい」を持つサービスを探します。サービスリンクロールに関するドキュメントをサービスで表示するには、リンクで [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 AWS Proton

AWS Proton は、AWSServiceRoleForProtonCodeBuildProvisioning 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。 AWS Proton CodeBuild プロビジョニングのサービスにリンクされたロール。

AWSServiceRoleForProtonCodeBuildProvisioning サービスにリンクされたロールは、次のサービスを信頼してロールを引き受けます。

- `codebuild.proton.amazonaws.com`

という名前のロールアクセス許可ポリシー`AWSProtonCodeBuildProvisioningServiceRolePolicy`により AWS Proton 、 は指定されたリソースに対して次のアクションを実行できます。

- アクション: CloudFormation スタックとトランスフォームの作成、管理、読み取り
- アクション: CodeBuild プロジェクトとビルドの作成、管理、読み取り

このポリシーの詳細については、「AWS マネージドポリシー: AWSProtonCodeBuildProvisioningServiceRolePolicy」を参照してください。

サービスリンクロールの作成、編集、削除を IAM エンティティ (ユーザー、グループ、ロールなど) に許可するにはアクセス許可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「サービスにリンクされた役割のアクセス許可」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 AWS Proton

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 マネジメントコンソール、AWS CLIまたは AWS API AWS Proton で で CodeBuild ベースのプロビジョニングを使用する環境を作成すると、 AWS Proton によってサービスにリンクされたロールが作成されます。

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。で CodeBuild ベースのプロビジョニングを使用する環境を作成すると AWS Proton、 はサービスにリンクされたロールを再度 AWS Proton 作成します。

のサービスにリンクされたロールの編集 AWS Proton

AWS Proton では、AWSServiceRoleForProtonCodeBuildProvisioning 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の作成後は、さまざまなエンティティがロールを参照す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 を使用してロールの説明を編集することはできます。詳細については、「IAM ユーザーガイド」の「サービスリンクロールの編集」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 AWS Proton

サービスリンクロールを必要とする機能やサービスが不要になった場合は、ロールを削除することをお勧めします。そうすることで、モニタリングや保守が積極的に行われていない未使用のエンティティを排除できます。ただし、手動で削除 AWS Proton する前に、 で CodeBuild ベースのプロビジョニングを使用するすべての環境とサービス (インスタンスとパイプライン) を削除する必要があります。

サービスリンク役割の手動による削除

IAM コンソール、 AWS CLI、または AWS API を使用して、AWSServiceRoleForProtonCodeBuildProvisioning サービスにリンクされた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。

AWS Proton サービスにリンクされたロールでサポートされているリージョン

AWS Proton は、サービス AWS リージョン が利用可能なすべての 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 全般のリファレンスの「AWS Proton エンドポイントとクォータ」を参照してください。

AWS Proton ID とアクセスのトラブルシューティング

次の情報は、 および IAM の使用時に発生する可能性がある一般的な問題の診断 AWS Proton と修正に役立ちます。

トピック

- でアクションを実行する権限がありません AWS Proton
- iam:PassRole を実行する権限がない
- 自分の 以外のユーザーに自分の AWS Proton リソース AWS アカウント へのアクセスを許可したい

でアクションを実行する権限がありません AWS Proton

にアクションを実行する権限がないと AWS マネジメントコンソール 通知された場合は、管理者に連絡してサポートを依頼する必要があります。管理者とは、サインイン認証情報を提供した担当者です。

以下のエラー例は、mateojackson IAM ユーザーがコンソールを使用して架空の *my-example-widget* リソースに関する詳細情報を表示しようとしているが、架空の proton:*GetWidget* 権限がないという場合に発生します。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 proton:GetWidget on resource: my-example-widget
```

この場合、Mateo は、proton:*GetWidget* アクションを使用して *my-example-widget* リソースにアクセスできるように、ポリシーの更新を管理者に依頼します。

iam:PassRole を実行する権限がない

iam:PassRole アクションを実行する権限がないというエラーが表示された場合は、ポリシーを更新して AWS Protonにロールを渡すことができるようにする必要があります。

一部の AWS のサービス では、新しいサービスロールまたはサービスにリンクされたロールを作成する代わりに、そのサービスに既存のロールを渡すことができます。そのためには、サービスにロールを渡す権限が必要です。

以下の例のエラーは、marymajor という IAM ユーザーがコンソールを使用して AWS Protonでアクションを実行しようとする場合に発生します。ただし、このアクションをサービスが実行するには、

サービスロールから付与された権限が必要です。Mary には、ロールをサービスに渡すアクセス許可がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

この場合、Mary のポリシーを更新してメアリーに `iam:PassRole` アクションの実行を許可する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン資格情報を提供した担当者が管理者です。

自分の 以外のユーザーに自分の AWS Proton リソース AWS アカウント へのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACL) をサポートするサービスの場合、それらのポリシーを使用して、リソースへのアクセスを付与できます。

詳細については、以下を参照してください:

- がこれらの機能 AWS Proton をサポートしているかどうかを確認するには、「」を参照してくださいが IAM と AWS Proton 連携する方法。
- 所有 AWS アカウント している のリソースへのアクセスを提供する方法については、「IAM ユーザーガイド」の「所有 AWS アカウント している別の の IAM ユーザーへのアクセスを提供する」を参照してください。
- リソースへのアクセスをサードパーティーに提供する方法については AWS アカウント、IAM ユーザーガイドの「サードパーティー AWS アカウント が所有する へのアクセスを提供する」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、IAM ユーザーガイド の 外部で認証されたユーザー (ID フェデレーション) へのアクセスの許可 を参照してください。
- クロスアカウントアクセスにおけるロールとリソースベースのポリシーの使用方法の違いについては、「IAM ユーザーガイド」の「IAM でのクロスアカウントのリソースへのアクセス」を参照してください。

での設定と脆弱性の分析 AWS Proton

AWS Proton は、お客様が提供したコードのパッチや更新を提供しません。お客様は、 で実行されているサービスやアプリケーションのソースコード、 AWS Proton サービスや環境テンプレートバンドルで提供されているコードなど、独自のコードにパッチを更新して適用する責任があります。

お客様は、環境およびサービス内のインフラストラクチャリソースを更新およびパッチ適用する責任があります。 AWS Proton は、リソースを自動的に更新またはパッチ適用しません。アーキテクチャ内のリソースのドキュメントを参照して、それぞれのパッチ適用ポリシーをご理解いただく必要があります。

お客様がリクエストした環境とサービスの更新をサービスおよび環境テンプレートのマイナーバージョンに提供する以外に、 AWS Proton は、お客様がサービスおよび環境テンプレートとテンプレートバンドルで定義するリソースにパッチや更新を提供しません。

詳細については、以下の リソースを参照してください。

- 責任共有モデル
- Amazon Web Services：セキュリティプロセスの概要

でのデータ保護 AWS Proton

AWS Proton は、データ保護に関する規制とガイドラインを含む AWS 責任共有モデルに準拠しています。 AWS は、すべての を実行する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。 は、このインフラストラクチャでホストされるデータの制御 AWS を維持します AWS のサービス。 お客様のコンテンツと個人データを処理するためのセキュリティ設定コントロールを含めます。 AWS のお客様および APN パートナー、 データコントローラーまたはデータ処理者として動作する は、 に入力した個人データについて責任を負います。 AWS クラウド

データ保護の目的で、 AWS アカウント 認証情報を保護し、 AWS Identity and Access Management (IAM) を使用して個々のユーザーアカウントを設定することをお勧めします。これにより、各ユーザーに各自の職務を果たすために必要なアクセス許可のみが付与されます。また、次の方法でデータを保護することもお勧めします:

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。TLS 1.2 以降が推奨されます。
- で API とユーザーアクティビティのログ記録を設定します AWS CloudTrail。

- AWS 暗号化ソリューションと、 内のすべてのデフォルトのセキュリティコントロールを使用します AWS のサービス。

ユーザーのアカウント番号などの機密の識別情報は、[Name (名前)] フィールドなどの自由形式のテキストフィールドに配置しないでください。これは、コンソール AWS Proton 、API、または SDK AWS のサービス を使用して AWS CLIまたは他の を操作する場合も同様です。 AWS SDKs リソース識別子の自由形式のテキストフィールドに入力したデータはすべて、診断ログの内容として取得される可能性があります。 AWS 外部サーバーへの URL を指定するときは、そのサーバーへのリクエストを検証するための認証情報を URL に含めないでください。

データ保護の詳細については、AWS セキュリティブログ のブログ投稿「AWS の責任共有モデルと GDPR」を参照してください。

サーバー側の保管データの暗号化

テンプレートバンドルを保存する S3 バケット内の保管中のテンプレートバンドル内の機密データを暗号化することを選択した場合は、SSE-S3 または SSE-KMS キーを使用して、登録済みの AWS Proton テンプレートにアタッチできるように AWS Proton がテンプレートバンドルを取得できるようにする必要があります。

転送中の暗号化

すべてのサービス間の通信は、SSL/TLS を使用して転送中に暗号化されます。

AWS Proton 暗号化キー管理

内では AWS Proton、すべての顧客データはデフォルトで AWS Proton 所有キーを使用して暗号化されます。カスタマー所有のマネージド AWS KMS キーを指定すると、次の段落で説明するように、すべてのカスタマーデータはカスタマー提供のキーを使用して暗号化されます。

AWS Proton テンプレートを作成するときは、キーを指定し、認証情報 AWS Proton を使用して、がキーを使用 AWS Proton できるようにする権限を作成します。

手動で許可を辞退したり、指定したキーを無効にするか削除した場合、 AWS Proton は指定されたキーで暗号化データを読み込めなくなり、`ValidationException` を破棄します。

AWS Proton 暗号化コンテキスト

AWS Proton は暗号化コンテキストヘッダーをサポートしています。暗号化コンテキストは、データに関する追加のコンテキスト情報を含めることができるキーと値のペアのオプションのセットです。

暗号化コンテキストの一般的な情報については、AWS Key Management Service デベロッパーガイドの「AWS Key Management Service の概念 - 暗号化コンテキスト」を参照してください。

暗号化コンテキストは、シークレットデータを含まない一連のキーと値のペアです。データを暗号化するリクエストに暗号化コンテキストを含めると、 AWS KMS は暗号化コンテキストを暗号論的に暗号化データにバインドします。データを復号するには、同じ暗号化コンテキストに渡す必要があります。

お客様は、暗号化コンテキストを使用して、監査レコードおよびログにおけるカスタマーマネージドキーの使用を識別できます。これは、 AWS CloudTrail や Amazon CloudWatch Logs などのログにもプレーンテキストで表示されます。

AWS Proton は、お客様指定または外部指定の暗号化コンテキストを取りません。

AWS Proton は、次の暗号化コンテキストを追加します。

```
{
  "aws:proton:template": "<proton-template-arn>",
  "aws:proton:resource": "<proton-resource-arn>"
}
```

最初の暗号化コンテキストは、リソースが関連付けられている AWS Proton テンプレートを識別し、カスタマーマネージドキーのアクセス許可と許可の制約としても機能します。

2 番目の暗号化コンテキストは、暗号化されている AWS Proton リソースを識別します。

次の例は、 AWS Proton 暗号化コンテキストの使用を示しています。

サービスインスタンスを作成するデベロッパー。

```
{
  "aws:proton:template": "arn:aws:proton:region_id:123456789012:service-template/my-
template",
  "aws:proton:resource": "arn:aws:proton:region_id:123456789012:service/my-service/
service-instance/my-service-instance"
}
```

テンプレートを作成する管理者。

```
{
```

```
  "aws:proton:template": "arn:aws:proton:region_id:123456789012:service-template/my-
template",
  "aws:proton:resource": "arn:aws:proton:region_id:123456789012:service-template/my-
template"
}
```

のインフラストラクチャセキュリティ AWS Proton

マネージドサービスである AWS Proton は、 AWS グローバルネットワークセキュリティで保護されています。 AWS セキュリティサービスと がインフラストラクチャ AWS 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環境を AWS 設計するには、「Security Pillar AWS Well-Architected Framework」の「Infrastructure Protection」を参照してください。

AWS が発行した API コールを使用して、ネットワーク AWS Proton 経由で にアクセスします。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 が必須で、TLS 1.3 をお勧めします。
- DHE (楕円ディフィー・ヘルマン鍵共有) や ECDHE (楕円曲線ディフィー・ヘルマン鍵共有) などの完全前方秘匿性 (PFS) による暗号スイート。これらのモードは Java 7 以降など、ほとんどの最新システムでサポートされています。

ネットワークの分離を向上させるには、次のセクションで説明 AWS PrivateLink するように を使用できます。

AWS Proton およびインターフェイス VPC エンドポイント (AWS PrivateLink)

VPC と の間にプライベート接続を確立するには、インターフェイス VPC エンドポイント AWS Proton を作成します。インターフェイスエンドポイントは、インターネットゲートウェイAWS PrivateLink、NAT デバイス、VPN 接続、または AWS Direct Connect 接続なしで AWS Proton APIs にプライベートにアクセスできるテクノロジーである を利用しています。VPC 内のインスタンスは、 AWS Proton APIs と通信するためにパブリック IP アドレスを必要としません。VPC と の間のトラフィック AWS Proton は Amazon ネットワークを離れません。

各インターフェイスエンドポイントは、サブネット内の 1 つ、または複数の Elastic Network Interface によって表されます。

詳細については、「Amazon VPC ユーザーガイド」の「インターフェイス VPC エンドポイント (AWS PrivateLink)」を参照してください。

AWS Proton VPC エンドポイントに関する考慮事項

のインターフェイス VPC エンドポイントを設定する前に AWS Proton、Amazon VPC ユーザーガイドのインターフェイスエンドポイントのプロパティと制限を確認してください。

AWS Proton は、VPC からのすべての API アクションの呼び出しをサポートしています。

VPC エンドポイントポリシーがサポートされています AWS Proton。デフォルトでは、 へのフルアクセス AWS Proton はエンドポイントを介して許可されます。詳細については、「Amazon VPC ユーザーガイド」の「VPC エンドポイントによるサービスのアクセスコントロール」を参照してください。

のインターフェイス VPC エンドポイントの作成 AWS Proton

Amazon VPC コンソールまたは AWS Command Line Interface () を使用して、 AWS Proton サービスの VPC エンドポイントを作成できますAWS CLI。詳細については、「Amazon VPC ユーザーガイド」のインターフェイスエンドポイントの作成を参照してください。

次のサービス名 AWS Proton を使用して、 用の VPC エンドポイントを作成します。

- com.amazonaws.*region*.proton

エンドポイントのプライベート DNS を有効にすると、 など、リージョンのデフォルトの DNS 名 AWS Proton を使用して に API リクエストを行うことができますproton.*region*.amazonaws.com。

詳細については、「Amazon VPC ユーザーガイド」の「インターフェイスエンドポイントを介したサービスへのアクセス」を参照してください。

の VPC エンドポイントポリシーの作成 AWS Proton

VPC エンドポイントには、 AWS Protonへのアクセスを制御するエンドポイントポリシーをアタッチできます。このポリシーでは、以下の情報を指定します。

- アクションを実行できるプリンシパル。
- 実行可能なアクション。

- アクションを実行できるリソース。

詳細については、「Amazon VPC ユーザーガイド」の「VPC エンドポイントでサービスへのアクセスを制御する」を参照してください。

例: AWS Proton アクションの VPC エンドポイントポリシー

以下は、 のエンドポイントポリシーの例です AWS Proton。エンドポイントにアタッチすると、このポリシーは、すべてのリソースのすべてのプリンシパルに対して、リストされた AWS Proton アクションへのアクセスを許可します。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Principal": "*",
      "Action": [
        "proton:ListServiceTemplates",
        "proton:ListServiceTemplateMajorVersions",
        "proton:ListServiceTemplateMinorVersions",
        "proton:ListServices",
        "proton:ListServiceInstances",
        "proton:ListEnvironments",
        "proton:GetServiceTemplate",
        "proton:GetServiceTemplateMajorVersion",
        "proton:GetServiceTemplateMinorVersion",
        "proton:GetService",
        "proton:GetServiceInstance",
        "proton:GetEnvironment",
        "proton:CreateService",
        "proton:UpdateService",
        "proton:UpdateServiceInstance",
        "proton:UpdateServicePipeline",
        "proton:DeleteService"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
```

```
}
```

でのログ記録とモニタリング AWS Proton

モニタリングは、 およびその他の AWS Proton AWS ソリューションの信頼性、可用性、パフォーマンスを維持する上で重要な部分です。 AWS には、実行中のインスタンスを監視し AWS Proton、問題が発生したときに報告し、必要に応じて自動アクションを実行するための以下のモニタリングツールが用意されています。

現時点では、 AWS Proton 自身は Amazon CloudWatch Logs または と統合されていません AWS Trusted Advisor。管理者は、CloudWatch を設定して使用して、サービステンプレートと環境テンプレートで定義されている AWS のサービス 他の をモニタリングできます。 AWS Proton は と統合されています AWS CloudTrail。

- Amazon CloudWatch は、 AWS リソースと で実行されるアプリケーションを AWS リアルタイムでモニタリングします。メトリクスの収集と追跡、カスタマイズしたダッシュボードの作成、および指定したメトリクスが指定したしきい値に達したときに通知またはアクションを実行するアラームの設定を行うことができます。例えば、CloudWatch で Amazon EC2 インスタンスの CPU 使用率などのメトリクスを追跡し、必要に応じて新しいインスタンスを自動的に起動できます。詳細については、「Amazon CloudWatch ユーザーガイド」を参照してください。
- Amazon CloudWatch Logs では、Amazon EC2 インスタンス、CloudTrail、その他ソースから得たログファイルのモニタリング、保存、およびアクセスが可能です。CloudWatch Logs は、ログファイル内の情報をモニタリングし、特定のしきい値が満たされたときに通知します。高い耐久性を備えたストレージにログデータをアーカイブすることも可能です。詳細については、「Amazon CloudWatch Logs ユーザーガイド」を参照してください。
- AWS CloudTrail は、 によって、または に代わって行われた API コールおよび関連イベントをキャプチャ AWS アカウント し、指定した Amazon S3 バケットにログファイルを配信します。呼び出し元のユーザーとアカウント AWS、呼び出し元の送信元 IP アドレス、呼び出しの発生日時を特定できます。詳細については、「AWS CloudTrail ユーザーガイド」を参照してください。
- Amazon EventBridge は、アプリケーションをさまざまなイベントソースのデータに簡単に接続できるようにするサーバーレスイベントバスサービスです。EventBridge は、独自のアプリケーション、Software-as-a-Service (SaaS) アプリケーションからリアルタイムデータのストリームを配信 AWS のサービス し、そのデータを Lambda などのターゲットにルーティングします。これにより、サービスで発生したイベントをモニタリングし、イベント駆動型アーキテクチャを構築できます。詳細については、EventBridge AWS Proton による自動化 および EventBridge ユーザーガイドを参照してください。

の耐障害性 AWS Proton

AWS グローバルインフラストラクチャは、 AWS リージョン およびアベイラビリティーゾーンを中心に構築されています。 AWS リージョンは、低レイテンシー、高スループット、高度に冗長なネットワークで接続された、物理的に分離および分離された複数のアベイラビリティーゾーンを提供します。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。

とアベイラビリティーゾーンの詳細については、 AWS リージョン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。

グローバル AWS インフラストラクチャに加えて、 AWS Proton はデータの耐障害性とバックアップのニーズをサポートする機能を提供します。

AWS Proton バックアップ

AWS Proton は、すべての顧客データのバックアップを維持します。完全に停止した場合、このバックアップを使用して、以前の有効な状態から AWS Proton および顧客データを復元できます。

のセキュリティのベストプラクティス AWS Proton

AWS Proton には、独自のセキュリティポリシーを開発および実装する際に考慮すべきセキュリティ機能が用意されています。以下のベストプラクティス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を説明するものではありません。これらのベストプラクティスはお客様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

- IAM を使用したアクセス制御
- テンプレートおよびテンプレートバンドルに認証情報を埋め込まない
- 暗号化を使用した機密データの保護
- AWS CloudTrail を使用して API コールを表示およびログに記録する

IAM を使用したアクセス制御

IAM は、ユーザーとそのアクセス許可を管理するために AWS のサービス 使用できる です AWS。で IAM を使用して AWS Proton 、テンプレート、環境、サービスの管理など、管理者と開発者が実行できる AWS Proton アクションを指定できます。IAM サービスロールを使用すると、 AWS Proton がユーザーに代わって他の サービスを呼び出すことができます。

AWS Proton および IAM ロールの詳細については、「」を参照してくださいの Identity and Access Management AWS Proton。

最小特権アクセスの実装 詳細については、『AWS Identity and Access Management ユーザーガイド』の 「IAM のポリシーとアクセス許可」を参照してください。

テンプレートおよびテンプレートバンドルに認証情報を埋め込まない

CloudFormation テンプレートやテンプレートバンドルに機密情報を埋め込むのではなく、スタックテンプレートで動的参照を使用することをお勧めします。

動的参照は、 AWS Systems Manager Parameter Store や などの他のサービスに保存および管理されている外部値を参照するコンパクトで強力な方法を提供します AWS Secrets Manager。動的なリファレンスを使用すると、CloudFormation は、スタックオペレーションおよび変更セットオペレーション中に指定されたリファレンスの値を取得して、その値を適切なリソースに渡します。ただし、CloudFormation が実際の参照値を保存することはありません。詳細については、CloudFormation ユーザーガイドの「動的リファレンスを使用してテンプレート値を指定する」を参照してください。

AWS Secrets Manager を使用して、データベースやその他のサービスの認証情報を安全に暗号化、保存、取得できます。AWS Systems Manager パラメータストア は、構成データ管理のための安全な階層型ストレージを提供します。

テンプレートパラメータの詳細については、CloudFormation ユーザーガイドの「https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/parameters-section-structure.html」を参照してください。

暗号化を使用した機密データの保護

内では AWS Proton、すべての顧客データはデフォルトで AWS Proton 所有キーを使用して暗号化されます。

プラットフォームチームのメンバーは、カスタマーマネージドキーを に提供 AWS Proton して、機密データを暗号化および保護できます。S3 バケットに保管中の機密データを暗号化します。詳細については、「でのデータ保護 AWS Proton」を参照してください。

AWS CloudTrail を使用して API コールを表示およびログに記録する

AWS CloudTrail は、 で API コールを行っているすべてのユーザーを追跡します AWS アカウント。API コールは、API、コンソール AWS Proton 、または AWS Proton AWS CLI コマンドを使用するたびにログに AWS Proton 記録されます。ログ記録を有効にして Amazon S3 バケットを指定し、ログを保存します。これにより、必要に応じて、アカウントでどのような AWS Proton 呼び出しを行ったかを監査できます。詳細については、「でのログ記録とモニタリング AWS Proton」を参照してください。

サービス間の混乱した代理の防止

混乱した代理問題は、アクションを実行するためのアクセス許可を持たないエンティティが、より特権のあるエンティティにアクションの実行を強制できてしまう場合に生じる、セキュリティ上の問題です。では AWS、サービス間のなりすましにより、混乱した代理問題が発生する可能性があります。サービス間でのなりすましは、1 つのサービス (呼び出し元サービス) が、別のサービス (呼び出し対象サービス) を呼び出すときに発生する可能性があります。呼び出し元サービスは、本来ならアクセスすることが許可されるべきではない方法でその許可を使用して、別のお客様のリソースに対する処理を実行するように操作される場合があります。これを防ぐために、 は、アカウント内のリソースへのアクセス権が付与されたサービスプリンシパルを持つすべてのサービスのデータを保護するのに役立つツール AWS を提供します。

リソースポリシーで aws:SourceArnおよび aws:SourceAccount グローバル条件コンテキストキーを使用して、 がリソースに別のサービス AWS Proton に付与するアクセス許可を制限することをお勧めします。aws:SourceArn の値に Amazon S3 バケット ARN などのアカウント ID が含まれていない場合は、両方のグローバル条件コンテキストキーを使用して、アクセス許可を制限する必要があります。同じポリシーステートメントでこれらのグローバル条件コンテキストキーの両方を使用し、アカウント ID にaws:SourceArn の値が含まれていない場合、aws:SourceAccount 値と aws:SourceArn 値の中のアカウントには、同じアカウント ID を使用する必要があります。クロスサービスのアクセスにリソースを 1 つだけ関連付けたい場合は、aws:SourceArn を使用します。そのアカウント内のリソースをクロスサービスの使用に関連付けることを許可する場合は、aws:SourceAccount を使用します。

の値は、 が AWS Proton 保存するリソースaws:SourceArnである必要があります。

混乱した代理問題から保護するための最も効果的な方法は、リソースの完全な ARN を指定して aws:SourceArn グローバル条件コンテキストキーを使用することです。リソースの完全な ARN が不明な場合や、複数のリソースを指定する場合は、aws:SourceArn グローバルコンテキスト条件キーを使用して、ARN の未知部分をワイルドカード (*) で表します。例えば、arn:aws::proton:*:*123456789012*:environment/* です。

次の例は、 で aws:SourceArnおよび aws:SourceAccount グローバル条件コンテキストキーを使用して、混乱した代理問題 AWS Proton を防ぐ方法を示しています。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Sid": "ExampleProtonConfusedDeputyPreventionPolicy",
        "Effect": "Allow",
        "Principal": {"Service": "proton.amazonaws.com"},
        "Action": "sts:AssumeRole",
        "Condition": {
            "StringEquals": {
                "aws:SourceAccount": "123456789012"
            },
            "ArnLike": {
                "aws:SourceArn": "arn:aws:proton:*:123456789012:environment/*"
            }
        }
    }
}
```

CodeBuild プロビジョニングカスタム Amazon VPC サポート

AWS Proton CodeBuild プロビジョニングは、 AWS Proton 環境アカウントにある CodeBuild プロジェクトで、お客様が指定した任意の CLI コマンドを実行します。これらのコマンドは通常、CDK などのコード Infrastructure as Code (IaC) ツールでリソースを管理します。Amazon VPC にリソースがあると、CodeBuild ではそれらにアクセスできないおそれがあります。この問題を回避するために、CodeBuild は特定の Amazon VPC 内で実行する機能をサポートしています。ユースケースの例を以下に示します。

- Python 用 PyPI、Java 用 Maven、Node.js 用 npm など、セルフホスト型の内部アーティファクトリポジトリから依存関係を取得する
- パイプラインを登録するとき、CodeBuild は、特定の Amazon VPC 内の Jenkins サーバーにアクセスする必要があります。
- アクセスルートが Amazon VPC エンドポイント経由に限定された設定の Amazon S3 バケット内のオブジェクトにアクセスする。
- プライベートサブネット上に分離された Amazon RDS データベース内のデータに対して、ビルドから統合テストを実行する。

詳細については、「CodeBuild と VPC のドキュメント」を参照してください。

CodeBuild プロビジョニングをカスタム VPC で実行する場合、 AWS Proton は簡単なソリューションを提供します。まず、環境テンプレートに、VPC ID、サブネット、セキュリティグループを追加します。次に、それらの値を環境仕様に入力します。以上で、特定の VPC をターゲットとする CodeBuild プロジェクトが自動的に作成されます。

環境テンプレートを更新する

Schema

VPC ID、サブネット、セキュリティグループは、環境仕様に組み込むために、テンプレートスキーマに追加する必要があります。

例 `schema.yaml`:

```
schema:
  format:
    openapi: "3.0.0"
  environment_input_type: "EnvironmentInputType"
  types:
    EnvironmentInputType:
      type: object
      properties:
        codebuild_vpc_id:
          type: string
        codebuild_subnets:
          type: array
          items:
            type: string
        codebuild_security_groups:
```

```
          type: array
          items:
            type: string
```

これで、マニフェストが使用する 3 つの新しいプロパティが追加されます。

- codebuild_vpc_id
- codebuild_subnets
- codebuild_security_groups

マニフェスト

CodeBuild で Amazon VPC 設定を行うとき、project_properties というオプションのプロパティをテンプレートマニフェストで使用できます。のコンテンツproject_propertiesは、CodeBuild プロジェクトを作成する CloudFormation スタックに追加されます。これにより、Amazon VPC CloudFormation プロパティだけでなく、ビルドタイムアウトなど、サポートされている CodeBuild CloudFormation プロパティも追加できます。proton-inputs.json に提供された同じデータが、project_properties の値でも利用できるようになります。

このセクションをお使いの manifest.yaml に追加してください。

```
project_properties:
          VpcConfig:
            VpcId: "{{ environment.inputs.codebuild_vpc_id }}"
            Subnets: "{{ environment.inputs.codebuild_subnets }}"
            SecurityGroupIds: "{{ environment.inputs.codebuild_security_groups }}"
```

manifest.yaml の結果は以下のように表示されます。

```
infrastructure:
  templates:
    - rendering_engine: codebuild
      settings:
        image: aws/codebuild/amazonlinux2-x86_64-standard:4.0
        runtimes:
          nodejs: 16
        provision:
          - npm install
          - npm run build
```

```
      - npm run cdk bootstrap
      - npm run cdk deploy -- --require-approval never
    deprovision:
      - npm install
      - npm run build
      - npm run cdk destroy -- --force
    project_properties:
      VpcConfig:
        VpcId: "{{ environment.inputs.codebuild_vpc_id }}"
        Subnets: "{{ environment.inputs.codebuild_subnets }}"
        SecurityGroupIds: "{{ environment.inputs.codebuild_security_groups }}"
```

環境の作成

CodeBuild プロビジョニング VPC 対応テンプレートで環境を作成するときは、Amazon VPC ID、サブネット、セキュリティグループを指定してください。

次のコマンドを実行して、リージョン内のすべての Amazon VPC ID のリストを取得します：

```
aws ec2 describe-vpcs
```

すべてのサブネット ID のリストを取得するには:

```
aws ec2 describe-subnets --filters "Name=vpc-id,Values=vpc-id"
```

> **Important**
>
> プライベートサブネットのみを含めます。パブリックサブネットを指定すると、CodeBuild は失敗します。パブリックサブネットにはインターネットゲートウェイへのデフォルトルートがありますが、プライベートサブネットにはありません。

セキュリティグループ ID を取得するには、次のコマンドを実行します。これらの ID は AWS マネジメントコンソールからも取得できます。

```
aws ec2 describe-security-groups --filters "Name=vpc-id,Values=vpc-id"
```

値は以下のようになります。

```
vpc-id: vpc-045ch35y28dec3a05
```

```
subnets:
  - subnet-04029a82e6ae46968
  - subnet-0f500a9294fc5f26a
 security-groups:
  - sg-03bc4c4ce32d67e8d
```

CodeBuild パーミッションの確認

Amazon VPC サポートには、Elastic Network Interface を作成する機能など、特定の権限が必要です。

コンソールで環境を作成する場合は、環境作成ウィザードでこれらの値を入力します。プログラムで環境を作成する場合、お使いの `spec.yaml` は以下のように表示されます。

```
proton: EnvironmentSpec

spec:
  codebuild_vpc_id: vpc-045ch35y28dec3a05
  codebuild_subnets:
    - subnet-04029a82e6ae46968
    - subnet-0f500a9294fc5f26a
  codebuild_security_groups:
    - sg-03bc4c4ce32d67e8d
```

AWS Proton リソースとタグ付け

AWS Proton Amazon リソースネーム (ARN) が割り当てられている リソースには、環境テンプレートとそのメジャーバージョンとマイナーバージョン、サービステンプレートとそのメジャーバージョンとマイナーバージョン、環境、サービス、サービスインスタンス、コンポーネント、リポジトリが含まれます。タグ付けすることによって、これらのリソースを整理して特定できます。タグを使用して、リソースを目的、所有者、環境、またはその他の基準で分類できます。詳細については、「タグ付け戦略」を参照してください。 AWS Proton リソースを追跡し、管理するには、次のセクションで説明するように、タグ付け機能を使用します。

AWS タグ付け

タグの AWS 形式でリソースにメタデータを割り当てることができます。各タグは、お客様が定義するキーとオプションの値で構成されます。タグは、リソースの管理、識別、整理、検索、フィルタリングに役立ちます。

> **Important**
>
> 個人情報 (PII) などの機密情報や秘匿性の高い情報はタグに追加しないようにします。タグには、請求など、多くの AWS のサービスからアクセスできます。タグは、プライベートデータまたは機密データに使用することを目的としたものではありません。

各タグは 2 つの部分で構成されます。

- タグキー (例: `CostCenter`、`Environment` または `Project`)。タグキーでは大文字と小文字が区別されます。
- タグ値 (オプション) (たとえば、`111122223333` または `Production`)。タグキーと同様に、タグ値では大文字と小文字が区別されます。

タグには、次の基本的な命名要件と使用要件が適用されます。

- 各リソースには、最大 50 個のユーザー作成タグを含めることができます。

> **Note**
>
> aws: プレフィックスで始まるシステム作成タグは AWS 使用のために予約されており、この制限にはカウントされません。aws: プレフィックスで始まるタグを編集または削除することはできません。

- タグキーは、リソースごとにそれぞれ一意である必要があります。また、各タグキーに設定できる値は 1 つのみです。
- タグキーは、UTF-8 で 1 ~ 128 文字の Unicode 文字である必要があります。
- UTF-8 では、タグ値は 1 文字以上で、最大 256 文字の Unicode 文字である必要があります。
- タグに使用できる文字は、UTF-8 対応の文字、数字、スペース、および _ . : / = + - @ です。

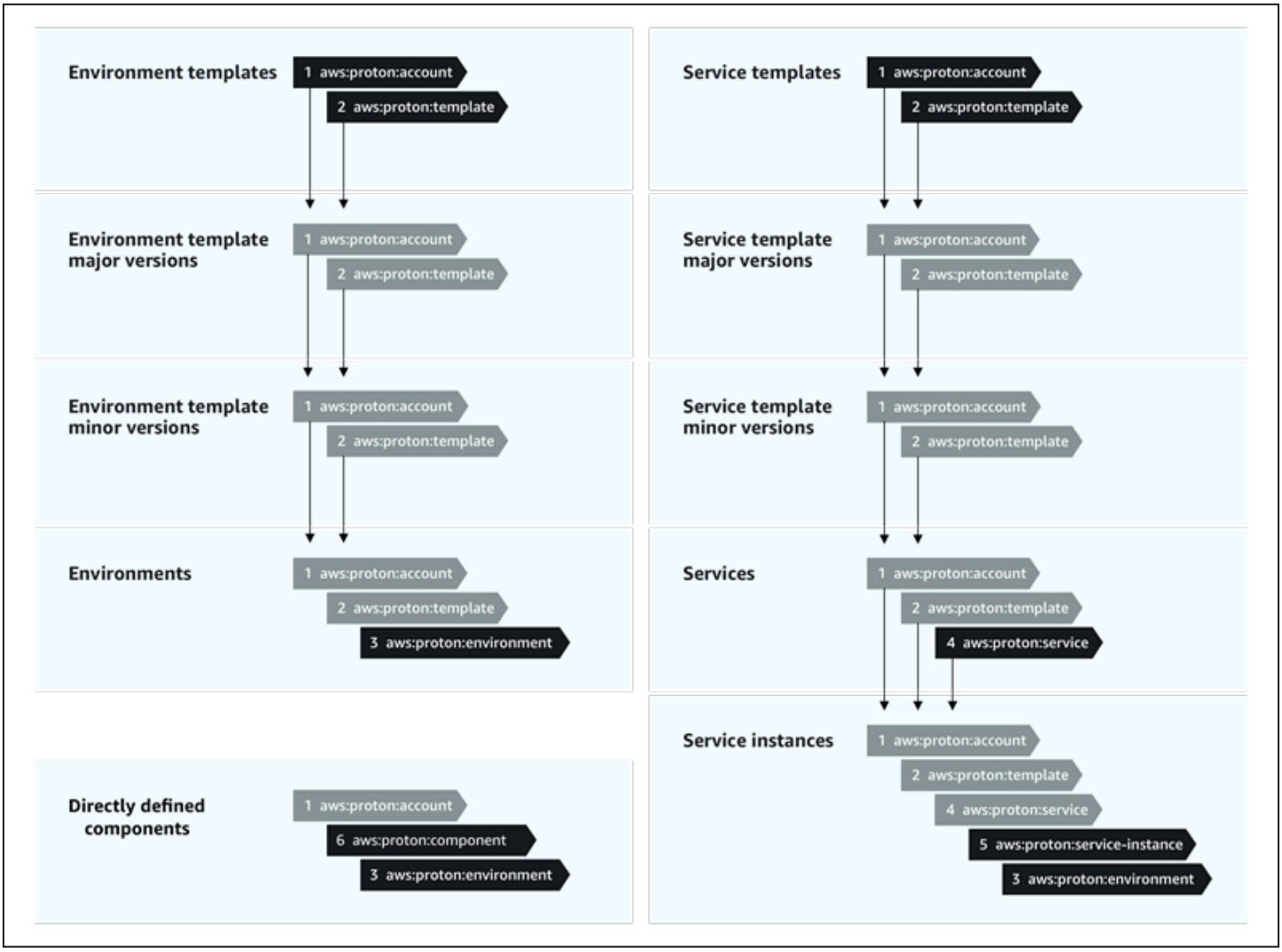
AWS Proton タグ付け

を使用すると AWS Proton、作成したタグと AWS Proton が自動的に生成するタグの両方を使用できます。

AWS Proton AWS マネージドタグ

AWS Proton リソースを作成すると、次の図に示すように、 は新しいリソースの AWS マネージドタグ AWS Proton を自動的に生成します。 AWS マネージドタグは、後で新しい AWS Proton リソースに基づく他のリソースに伝播されます。たとえば、環境テンプレートのマネージドタグはそのバージョンに伝えられ、サービスのマネージドタグはサービスインスタンスに伝えられます。

> **Note**
>
> AWS マネージドタグは、環境アカウント接続では生成されません。詳細については、「the section called “アカウント接続”」を参照してください。



プロビジョニングされたリソースへのタグの伝達

サービステンプレートや環境テンプレートで定義されているようなプロビジョニングされたリソースが AWS タグ付けをサポートしている場合、 AWS マネージドタグはカスタマーマネージドタグとしてプロビジョニングされたリソースに伝播されます。これらのタグは、 AWS タグ付けをサポートしていないプロビジョニングされたリソースには伝達されません。

AWS Proton は、次の表に示すように、 AWS Proton アカウント、登録されたテンプレート、デプロイされた環境、サービスとサービスインスタンスによってリソースにタグを適用します。 AWS マネージドタグを使用して AWS Proton リソースを表示および管理することはできますが、変更することはできません。

AWS マネージドタグキー	伝達されたユーザーマネージドキー	説明
aws:proton:account	proton:account	AWS Proton リソースを作成およびデプロイする AWS アカウント。
aws:proton:template	proton:template	選択したテンプレートの ARN。
aws:proton:environment	proton:environment	選択した環境の ARN。
aws:proton:service	proton:service	選択したサービスの ARN。
aws:proton:service-instance	proton:service-instance	選択したサービスインスタンスの ARN。
aws:proton:component	proton:component	選択したコンポーネントの ARN。

以下は、 AWS Proton リソースの AWS マネージドタグの例です。

```
"aws:proton:template" = "arn:aws:proton:region-id:account-id:environment-template/env-template"
```

以下は、 マネージドタグから伝播されたプロビジョニングされたリソースに適用されるカスタマー AWS マネージドタグの例です。

```
"proton:environment:database" = "arn:aws:proton:region-id:account-id:rds/env-db"
```

AWSマネージドプロビジョニングでは、 は伝播されたタグをプロビジョニングされたリソースに直接 AWS Proton 適用します。

セルフマネージドプロビジョニングでは、 AWS Proton は、プロビジョンプルリクエスト (PR) で送信するレンダリングされた IaC ファイルとともに、伝播されたタグを利用できるようにします。タグは文字列マップ変数 proton_tags で提供されます。に AWS Proton タグを含めるに

は、Terraform 設定でこの変数を参照することをお勧めしますdefault_tags。これにより、 AWS Proton タグはプロビジョニングされたすべてのリソースに伝えられます。

次の例は、環境 Terraform テンプレートにおけるこのタグ伝達方法を示しています。

proton_tags 変数の定義は次のとおりです。

proton.environment.variables.tf:

```
variable "environment" {
  type = object({
    inputs = map(string)
    name = string
  })
}

variable "proton_tags" {
  type = map(string)
  default = null
}
```

この変数にタグ値を割り当てる方法は次のとおりです。

proton.auto.tfvars.json:

```
{
  "environment": {
    "name": "dev",
    "inputs": {
      "ssm_parameter_value": "MyNewParamValue"
    }
  }

  "proton_tags" : {
    "proton:account" : "123456789012",
    "proton:template" : "arn:aws:proton:us-east-1:123456789012:environment-template/
fargate-env",
    "proton:environment" : "arn:aws:proton:us-east-1:123456789012:environment/dev"
  }
}
```

また、プロビジョニングされたリソースに追加されるように Terraform 設定に AWS Proton タグを追加する方法は次のとおりです。

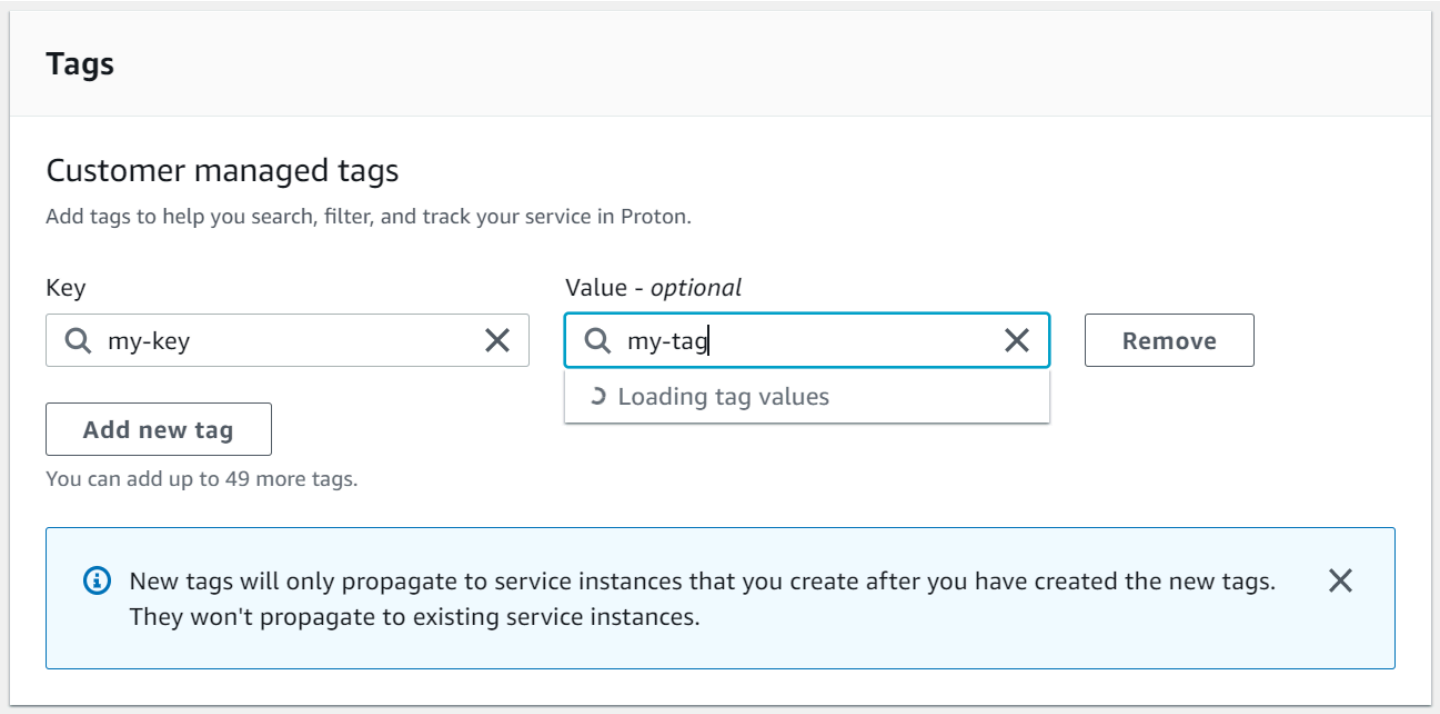
```
# Configure the AWS Provider
provider "aws" {
  region = var.aws_region
  default_tags {
    tags = var.proton_tags
  }
}
```

ユーザーマネージドタグ

各 AWS Proton リソースには、カスタマーマネージドタグの最大クォータが 50 個あります。カスタマーマネージドタグは、 AWS マネージドタグと同じ方法で子 AWS Proton リソースに伝達されます。ただし、既存のリソースやプロビジョニングされた AWS Proton リソースには伝達されません。既存の子リソースを持つ リソースに AWS Proton 新しいタグを適用し、既存の子リソースに新しいタグを付ける場合は、 コンソールまたは を使用して、既存の各子リソースに手動でタグを付ける必要があります AWS CLI。

コンソールおよび CLI を使用してタグを作成する

コンソールを使用して AWS Proton リソースを作成すると、次のコンソールスナップショットに示すように、作成手順の 1 ページ目または 2 ページ目にカスタマーマネージドタグを作成する機会が与えられます。[Add new tag (新しいタグを追加)] を選択し、キーと値を入力して続行します。



AWS Proton コンソールを使用して新しいリソースを作成したら、詳細ページから AWS マネージドタグとカスタマーマネージドタグのリストを表示できます。

タグの作成または編集

1. AWS Proton コンソールで、タグのリストを表示できる AWS Proton リソースの詳細ページを開きます。
2. [Manage tags (タグの管理)] を選択します。
3. [Managed tags (タグの管理)] ページでは、タグを表示、作成、削除、および編集できます。上部にリストされている AWS マネージドタグは変更できません。ただし、カスタマー管理タグにを追加したり変更したりするには、 AWS 管理タグの後にリストされている編集フィールドを使用します。

   新しいタグを追加するには、[Create Tag (タグの作成)] を選択します。
4. 新しいタグのキーと値を入力します。
5. タグを編集するには、選択したキーのタグ値フィールドに値を入力します。
6. タグを削除するには、削除したいタグを選択して [Remove (削除)] を選択します。
7. 変更を加え終わったら、[Save changes (変更の保存)] を選択します。

を使用してタグを作成する AWS Proton AWS CLI

を使用してタグを表示、作成、削除、編集できます AWS Proton AWS CLI。

次の例に示すように、リソースのタグを作成または編集できます。

```
$ aws proton tag-resource \
    --resource-arn "arn:aws:proton:region-id:account-id:service-template/webservice" \
    --tags '[{"key":"mykey1","value":"myval1"},{"key":"mykey2","value":"myval2"}]'
```

次の例に示すように、リソースのタグを削除できます。

```
$ aws proton untag-resource \
    --resource-arn "arn:aws:proton:region-id:account-id:service-template/webservice" \
    --tag-keys '["mykey1","mykey2"]'
```

最後の例に示すように、リソースのタグを一覧表示できます。

```
$ aws proton list-tags-for-resource \
```

```
    --resource-arn "arn:aws:proton:region-id:account-id:service-template/webservice"
```

トラブルシューティング AWS Proton

に関する問題のトラブルシューティングについて説明します AWS Proton。

トピック

CloudFormation 動的パラメータを参照するデプロイエラー

CloudFormation 動的変数を参照するデプロイエラーが表示された場合、それらが Jinja エスケープされていることを確認してください。これらのエラーは、動的変数の Jinja の誤解釈に起因する可能性があります。CloudFormation 動的パラメータ構文は、 AWS Proton パラメータで使用する Jinja 構文と非常によく似ています。

CloudFormation 動的変数の構文の例:

`'{{resolve:secretsmanager:MySecret:SecretString:password:EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE}}'`.

AWS Proton パラメータ Jinja 構文の例：

`'{{ service_instance.environment.outputs.env-outputs }}'`.

こういった誤解釈エラーを回避するには、以下の例に示すように、CloudFormation 動的パラメータをエスケープします。

この例は CloudFormation ユーザーガイドのものです。 AWS Secrets Manager secret-name セグメントと json-key セグメントを使用して、シークレットに保存されているサインイン認証情報を取得できます。

```
MyRDSInstance:
  Type: AWS::RDS::DBInstance
  Properties:
    DBName: 'MyRDSInstance'
      AllocatedStorage: '20'
      DBInstanceClass: db.t2.micro
      Engine: mysql
      MasterUsername: '{{resolve:secretsmanager:MyRDSSecret:SecretString:username}}'
```

```
      MasterUserPassword:
 '{{resolve:secretsmanager:MyRDSSecret:SecretString:password}}'
```

CloudFormation の動的パラメータをエスケープする方法には 2 とおりがあります。

- ブロックを {% raw %} and {% endraw %} で囲む:

```
'{% raw %}'
MyRDSInstance:
  Type: AWS::RDS::DBInstance
  Properties:
    DBName: 'MyRDSInstance'
      AllocatedStorage: '20'
      DBInstanceClass: db.t2.micro
      Engine: mysql
      MasterUsername: '{{resolve:secretsmanager:MyRDSSecret:SecretString:username}}'
      MasterUserPassword:
 '{{resolve:secretsmanager:MyRDSSecret:SecretString:password}}'
'{% endraw %}'
```

- パラメータを "{{ }}" で囲む:

```
MyRDSInstance:
  Type: AWS::RDS::DBInstance
  Properties:
    DBName: 'MyRDSInstance'
      AllocatedStorage: '20'
      DBInstanceClass: db.t2.micro
      Engine: mysql
      MasterUsername:
 "{{ '{{resolve:secretsmanager:MyRDSSecret:SecretString:username}}' }}"
      MasterUserPassword:
 "{{ '{{resolve:secretsmanager:MyRDSSecret:SecretString:password}}' }}"
```

詳細については、「Jinja escaping」(Jinja エスケープ処理) を参照してください。

AWS Proton クォータ

次の表に AWS Proton 、クォータを示します。すべての値は、 AWS アカウントごと、サポートされている AWS リージョンごとです。

リソースクォータ	デフォルトの制限	引き上げ可能?
テンプレートバンドルの最大サイズ	10 MB	✗いいえ
テンプレートマニフェストファイルの最大サイズ	2 MB	✗いいえ
テンプレートスキーマファイルの最大サイズ	2 MB	✗いいえ
テンプレートスキーマファイルの最大サイズ	2 MB	✗いいえ
各テンプレート名の最大長	100 文字	✗いいえ
バンドルごとの CloudFormation テンプレートファイルの最大数	1	✗いいえ
アカウント、サービス、環境テンプレートの組み合わせごとの登録テンプレートの最大数	1,000	✓ はい
テンプレートごとの登録テンプレートバージョンの最大数	1,000	✓ はい
CodeBuild プロビジョニングバンドルごとの最大ファイル数	500	✗いいえ
アカウントごとの環境の最大数	1,000	✓ はい
アカウントごとのサービスの最大数	1,000	✓ はい
サービスごとのサービスインスタンスの最大数	20	✓ はい
アカウントごとのコンポーネントの最大数	1,000	✓ はい
環境アカウントごとの環境アカウント接続の最大数	1,000	✓ はい

ドキュメント履歴

次の表に、 の最新リリース AWS Proton とお客様からのフィードバックに関連するドキュメントの重要な変更点を示します。このドキュメントの更新に関する通知を受け取るには、RSS フィードにサブスクライブできます。

- API バージョン: 2020-07-20

変更	説明	日付
サポート終了通知	2026 年 10 月 7 日、 AWS はのサポートを終了します AWS Proton。	2025 年 10 月 7 日
マネージドポリシーの更新	AWSProtonCodeBuildProvisioningServiceRolePolicy ポリシーを更新しました。	2024 年 6 月 15 日
マネージドポリシーの更新	AWSProtonDeveloperAccess ポリシーを更新しました。	2024 年 4 月 25 日
マネージドポリシーの更新	AWSProtonFullAccess ポリシーを更新しました。	2024 年 4 月 25 日
マネージドポリシーの更新	AWSProtonSyncServiceRolePolicy ポリシーを更新しました。	2024 年 4 月 25 日
マネージドポリシーの更新	AWSProtonCodeBuildProvisioningServiceRolePolicy ポリシーを更新しました。	2023 年 5 月 12 日

サービス同期設定。	AWS Proton は、サービス同期設定のサポートを追加します。	2023 年 3 月 31 日
CodeBuild	AWS Proton は CodeBuild プロビジョニングのサポートを追加します。	2022 年 11 月 16 日
マネージドポリシーの更新	CodeBuild プロビジョニング用のビルドを実行するために必要なアクセス許可を AWS Proton CodeBuild に付与する AWSProtonCodeBuildProvisioningBasicAccess ポリシーを追加しました。	2022 年 11 月 11 日
Terraform タグの伝達	「タグ付け」の章に Terraform タグの伝達を追加しました。	2022 年 9 月 16 日
API 移行ガイド	GA 以前の API 移行ガイドを削除しました。	2022 年 8 月 12 日
AWS Proton オブジェクト	AWS Proton オブジェクトとその他のオブジェクト AWS やサードパーティーオブジェクトとの関係に関するトピックを追加しました。「AWS Proton オブジェクト」を参照してください。	2022 年 7 月 29 日
リンク先リポジトリの説明	リンクされた (登録された) リポジトリの目的と使用方法をガイド全体で明確にしました。	2022 年 7 月 18 日

ガイドマージ	2 つの個別の管理者ガイドとユーザーガイドを 1 つのAWS Proton ユーザーガイドに統合しました。	2022 年 6 月 30 日
マネージドポリシーの更新	新しい AWS Proton API オペレーションへのアクセスを提供し、一部の AWS Proton コンソールオペレーションのアクセス許可の問題を修正するために、 管理ポリシーを更新しました。の AWS マネージドポリシーを参照してください AWS Proton。	2022 年 6 月 20 日
CLI の開始	「Getting started with the AWS CLI」 を更新し、新しいテンプレートライブラリを使用する新しいチュートリアルを追加しました。	2022 年 6 月 14 日
直接定義されたコンポーネント	「コンポーネント」の章を追加し、ガイド全体で関連する変更を加えました。	2022 年 6 月 1 日
AWS Proton テンプレートライブラリ	AWS Proton テンプレートライブラリトピックを追加しました。	2022 年 5 月 6 日
Terraform 一般提供 (GA)	プルリクエストプロビジョニングの名前をセルフマネージドプロビジョニングに変更しました。「プロビジョニング方法」トピックを追加しました。	2022 年 3 月 23 日

リポジトリのタグ付け	リポジトリリソースのタグ付けのサポートを追加しました。「リポジトリへのリンクの作成」を参照してください。	2022 年 3 月 23 日
ドキュメントの更新	環境アカウント接続のタグ付けを追加しました。	2021 年 11 月 26 日
テンプレート同期と Terraform プレビュー	一般提供 (GA) バージョン向けとして、プレビューに。テンプレートの同期機能によるテンプレートの自動バージョニングと「Terraform によるプルリクエストプロビジョニング」を追加しました。API 移行ガイドが復活しました。	2021 年 11 月 24 日
ドキュメントの更新	EventBridge チュートリアル、入門ワークフロー、AWS Proton の仕組み、テンプレートバンドルセクションの機能強化を追加しました。	2021 年 9 月 17 日
AWS Proton コンソールヘルプパネルのリリース	コンソールにヘルプパネルを追加しました。コンソールテンプレートバージョンの削除で下位バージョンが削除されなくなりました。API 移行ガイドを削除しました。	2021 年 9 月 8 日
AWS Proton 一般提供 (GA) リリース	クロスアカウント環境、EventBridge モニタリング、IAM 条件キー、冪等性サポート、およびクォータ増加を追加しました。	2021 年 6 月 9 日

サービスのサービスインスタンスを追加および削除し、 を使用する環境に既存の外部インフラストラクチャを使用する AWS Proton	このパブリックプレビューリリースには、サービスへのサービスインスタンスの追加と削除、AWS Proton 環境内の既存の外部インフラストラクチャの使用、環境、サービスインスタンス、パイプラインのデプロイのキャンセルを可能にする更新が含まれています。 は PrivateLink をサポートする AWS Proton ようになりました。リソースで使用中のマイナーバージョンを誤って削除しないように、付加的な削除検証を追加しました。	2021 年 4 月 27 日
を使用したタグ付け AWS Proton	パブリックプレビューリリース 2 には AWS Proton 、タグ付けと、サービスパイプラインなしでサービスを起動する機能が含まれています。	2021 年 3 月 5 日
初回リリース	一部のリージョンで公開プレビューリリースが提供されるようになりました。	2020 年 12 月 1 日

AWS 用語集

最新の AWS 用語については、 AWS の用語集 リファレンスのAWS 用語集を参照してください。